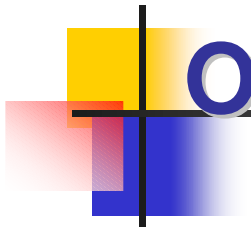
The ARM Architecture

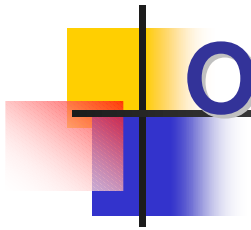
P. Bakowski

bako@ieee.org



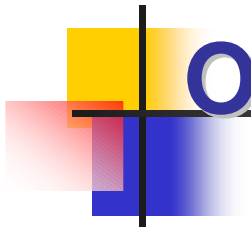
Outline

- ARM history
- ARM architecture
- ARM ISA features
- ISA extensions
- Architecture implementations



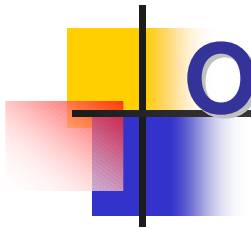
Outline

- ARM history
- ARM architecture
- ARM ISA features
- ISA extensions
- Architecture implementations



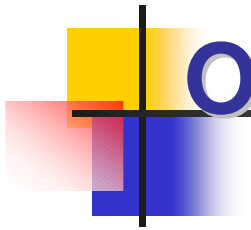
Outline

- ARM history
- ARM architecture
- ARM ISA features (Instruction Set Architecture)
- ISA extensions
- Architecture implementations



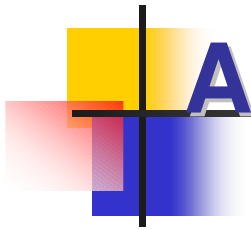
Outline

- ARM history
- ARM architecture
- ARM ISA features
- ISA extensions
- Architecture implementations



Outline

- ARM history
- ARM architecture
- ARM ISA features
- ISA extensions
- Architecture implementations



ARM pre-history

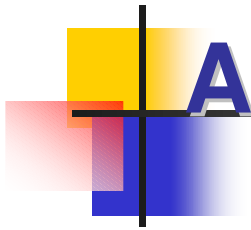
- Started in 1983 – as ACORN Ltd

designing new architecture after the refusal of using 80286

ARM pre-history

- Started in 1983 – as ACORN Ltd

  designing new architecture after the refusal of using 80286



ARM pre-history

- Started in 1983 – as ACORN Ltd

  designing new architecture after the refusal of using 80286

  - ARMv1 – 1985 based on RISC 1

ARM pre-history

- Started in 1983 – as ACORN Ltd

  designing new architecture after the refusal of using 80286

  - ARMv1 – 1985 based on RISC 1
  - ARMv2 – 32-bit data; 26-bit address

    simple RISC architecture, 30 K transistors, no microcode, no cache

Advanced RISC Machines

- Advanced Risc Machines - 1990

  spun off as separate company working with Apple on newer versions of core

- ARM6 created
  - basis for Apple Newton PDA
  - full 32-bit CPU with multiplication

Advanced RISC Machines

- Advanced Risc Machines - 1990

  spun off as separate company working with Apple on newer versions of core

- ARM6 created
  - basis for Apple Newton PDA
  - full 32-bit CPU with multiplication

Advanced RISC Machines

- Advanced Risc Machines - 1990

  spun off as separate company working with Apple on newer versions of core

- ARM6 created – implementation of ARMv2
  - basis for Apple Newton PDA
  - full 32-bit CPU with multiplication

Advanced RISC Machines

- Advanced Risc Machines - 1990

  spun off as separate company working with Apple on newer versions of core

- ARM6 created
  - basis for Apple Newton PDA
  - full 32-bit CPU with multiplication

Advanced RISC Machines

- Advanced Risc Machines - 1990

  spun off as separate company working with Apple on newer versions of core

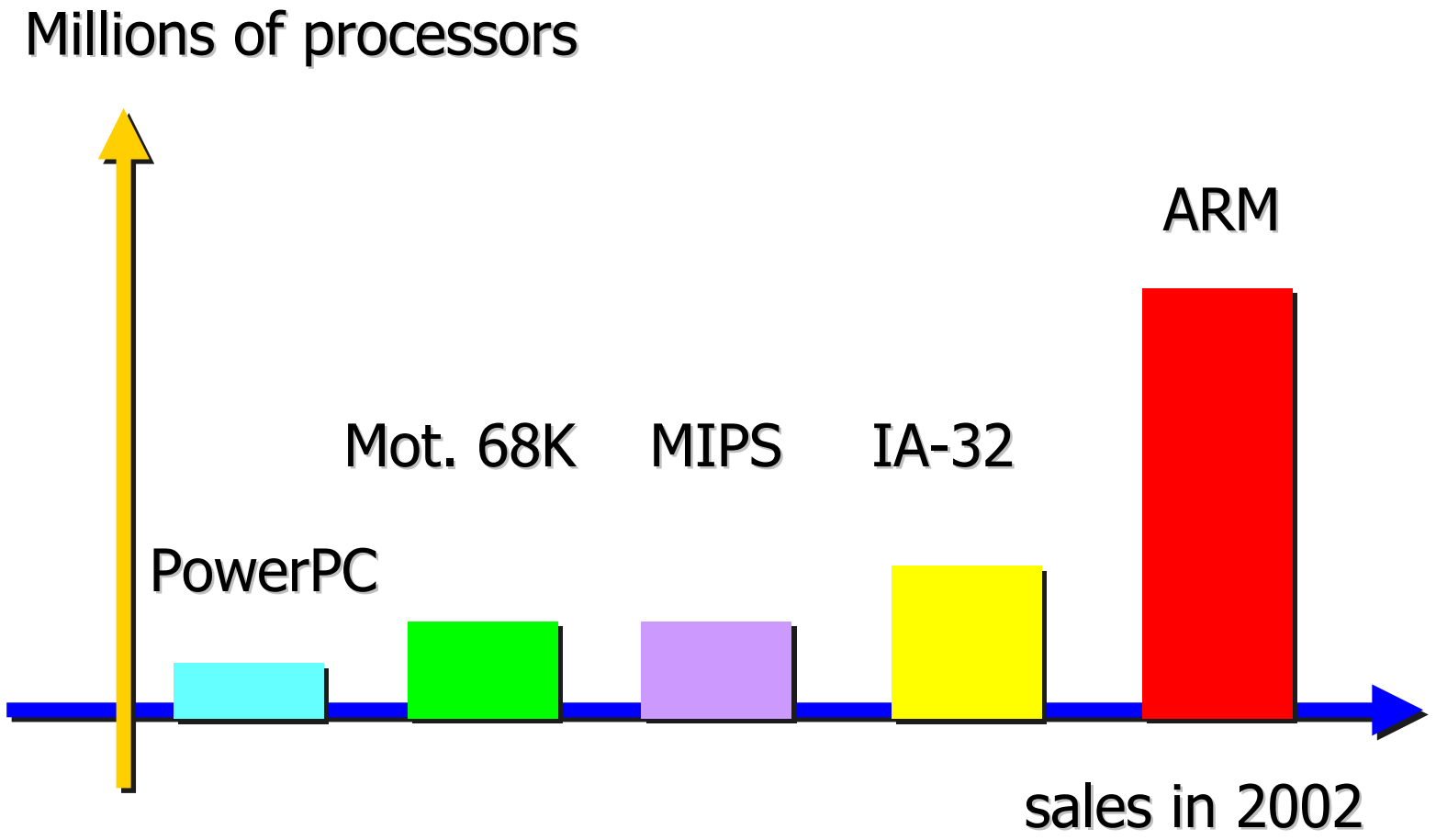
- ARM6 created
  - basis for Apple Newton PDA
  - full 32-bit CPU with multiplication

ARM place in the market

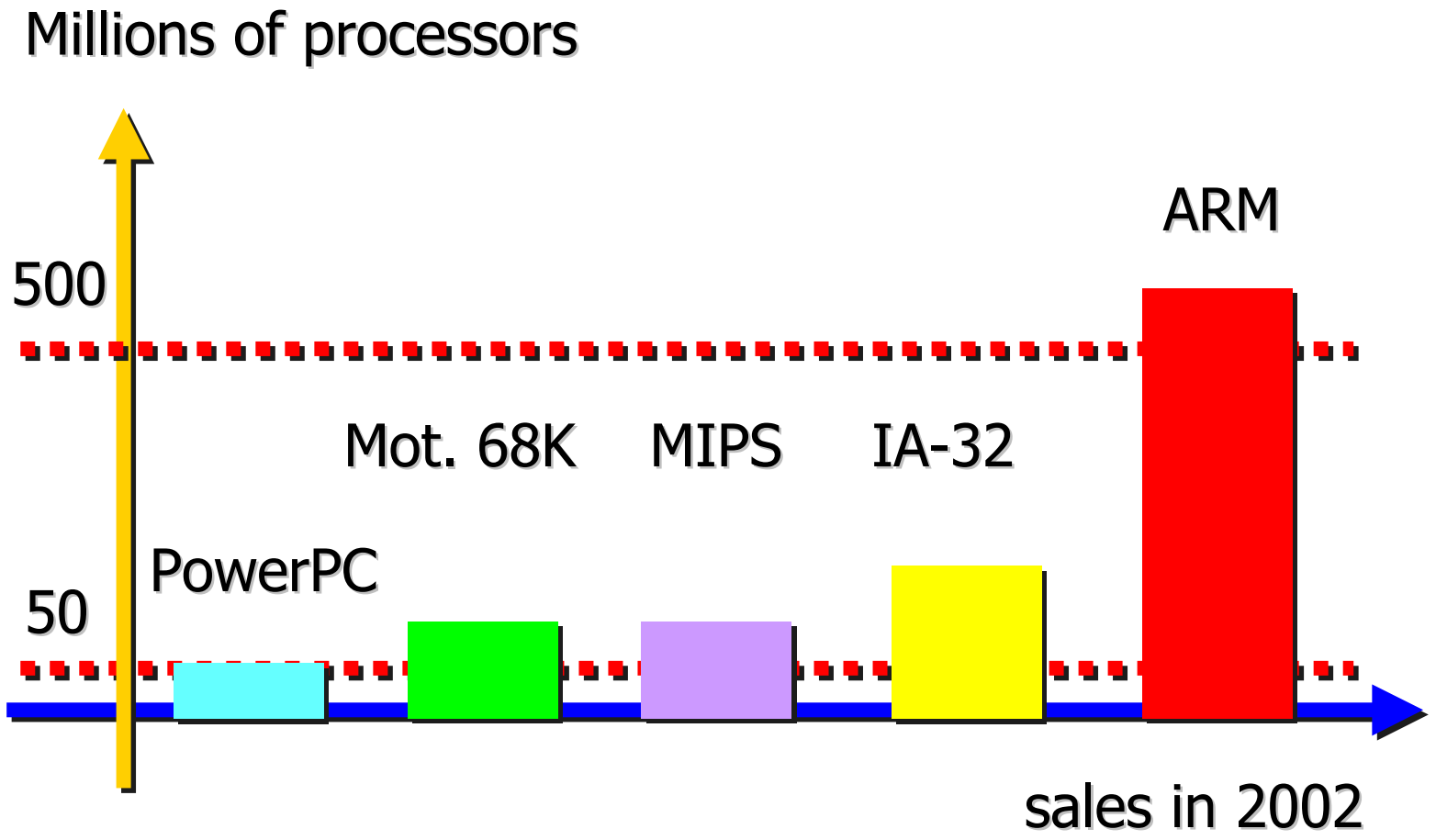
ARM is leading provider of 32-bit embedded RISC microprocessors:

- common architecture: compatible versions
- high performance
- low power consumption
- low system cost

ARM place on the market of µP



ARM place on the market of µP

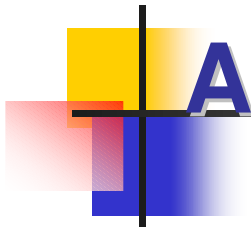


ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network

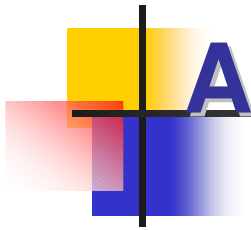
- Secure applications – smartcards and SIMs
- Open platforms running complex OS



ARM applications

Embedded real-time systems for:

- Mass storage
  - Automotive
  - Industrial
  - Network
- Secure applications – smartcards and SIMs
- Open platforms running complex OS

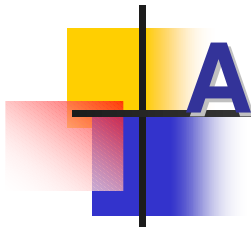


ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network

- Secure applications – smartcards and SIMs
- Open platforms running complex OS

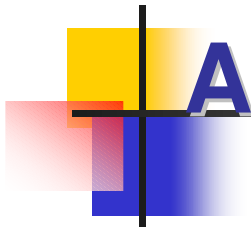


ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network

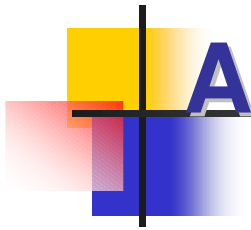
- Secure applications – smartcards and SIMs
- Open platforms running complex OS



ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network
- Secure applications – smartcards and SIMs
- Open platforms running complex OS

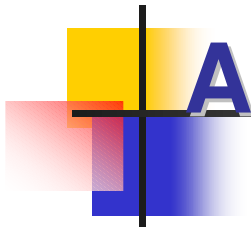


ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network

- Secure applications – smartcards and SIMs
- Open platforms running complex OS



ARM applications

Embedded real-time systems for:

- Mass storage
- Automotive
- Industrial
- Network

- Secure applications – smartcards and SIMs
- Open platforms running complex OS

Licensing ARM technology

Implementation license

- most popular
- hard or soft core (macro cells)
- complete information to design & manufacture integrating circuits containing ARM core
- plan to be used in several products

Licensing ARM technology

Implementation license

- most popular
  - hard or soft core (macro cells)
  - complete information to design & manufacture integrating circuits containing ARM core
  - plan to be used in several products

Licensing ARM technology

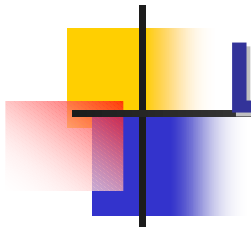
Implementation license

- most popular
- hard or soft core (macro cells)
- complete information to design & manufacture integrating circuits containing ARM core
- plan to be used in several products

Licensing ARM technology

Implementation license

- most popular
- hard or soft core (macro cells)
- complete information to design & manufacture integrating circuits containing ARM core
- plan to be used in several products



Licensing ARM technology

Implementation license

- most popular
- hard or soft core (macro cells)
- complete information to design & manufacture integrating circuits containing ARM core
- plan to be used in several products

Licensing ARM technology

Foundry license

- for fab-less semiconductor vendors
- to develop & sell ARM core-based products manufactured by licensed companies (foundries)

Licensing ARM technology

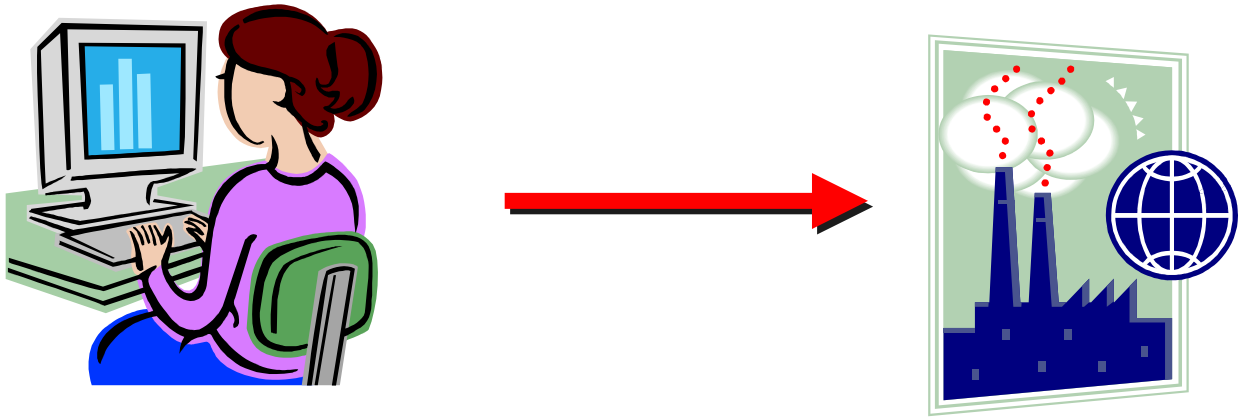
Foundry license

- for fab-less semiconductor vendors
  - to develop & sell ARM core-based products manufactured by licensed companies (foundries)

Licensing ARM technology

Foundry license

- for fab-less semiconductor vendors
- to develop & sell ARM core-based products manufactured by licensed companies (foundries)



Licensing ARM technology

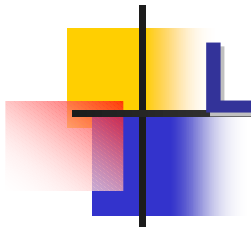
Architecture license

- to develop own CPU implementations
  - INTEL
  - SAMSUNG
  - TI

Licensing ARM technology

Architecture license

- to develop own CPU implementations
  - INTEL
  - SAMSUNG
  - TI



Licensing ARM technology

Architecture license

- to develop own CPU implementations
  - INTEL
  - SAMSUNG
  - TI

Licensing ARM technology

Academic license

- basic building blocks of the core to allow simulation and design of prototypes parts for academic research
- enables a core simulation environment to be created

Licensing ARM technology

Academic license

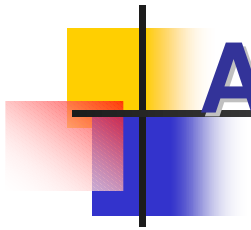
- basic building blocks of the core to allow simulation and design of prototypes parts for academic research
- enables a core simulation environment to be created

Licensing ARM technology

Academic license

- basic building blocks of the core to allow simulation and design of prototypes parts for academic research

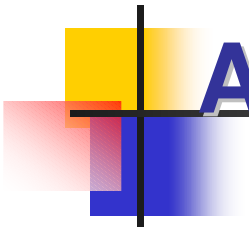
- enables a core simulation environment to be created



Architecture definition

The rules for how the microprocessor will behave:

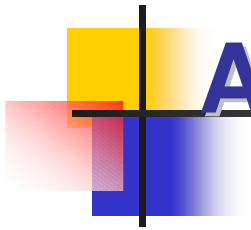
- instruction set specification (ISA)
- programming model
- operating system interface
- specification for the external interface



Architecture definition

The rules for how the microprocessor will behave:

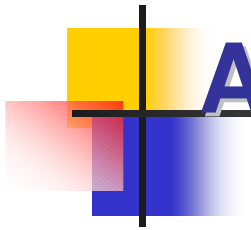
- instruction set specification (ISA)
- programming model
- operating system interface
- specification for the external interface



Architecture definition

The rules for how the microprocessor will behave:

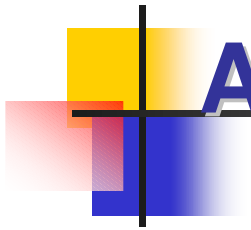
- instruction set specification (ISA)
- programming model: registers, flags, ..
- operating system interface
- specification for the external interface



Architecture definition

The rules for how the microprocessor will behave:

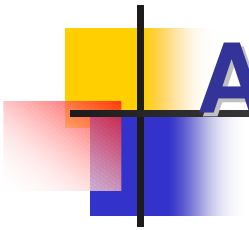
- instruction set specification (ISA)
- programming model
- operating system interface: modes, MMU, ..
- specification for the external interface



Architecture definition

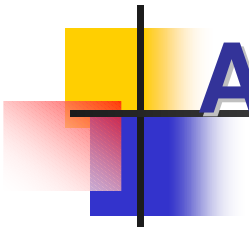
The rules for how the microprocessor will behave:

- instruction set specification (ISA)
- programming model
- operating system interface
- specification for the external interface



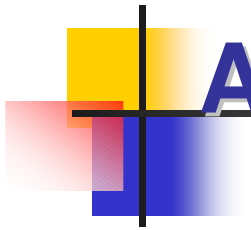
ARM architecture versions

- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip



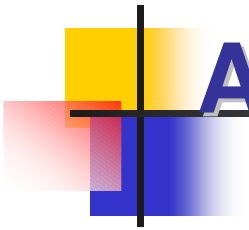
ARM architecture versions

- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip



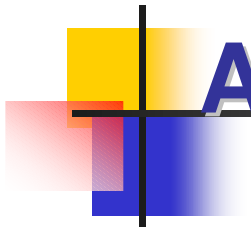
ARM architecture versions

- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip



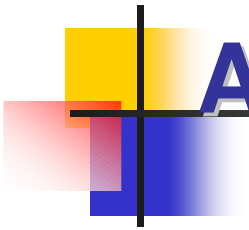
ARM architecture versions

- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip



ARM architecture versions

- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip



ARM architecture versions

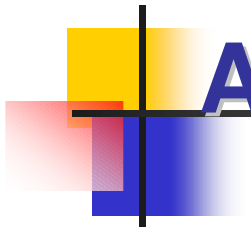
- ARMv1
  - first version of ARM processor
  - 26-bit addressing, no multiply, no coprocessor
- ARMv2
  - included 32-bit result multiply, coprocessor
  - ARM2 (implementation): first commercial chip

ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent

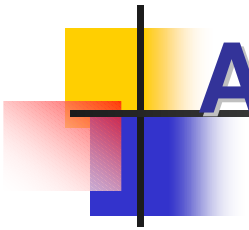
ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent



ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent

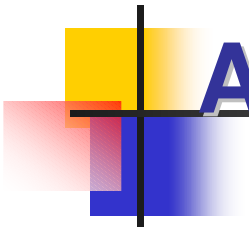


ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent

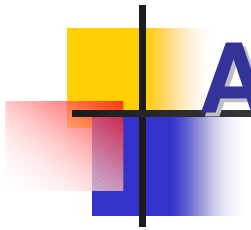
ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent



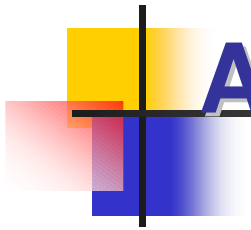
ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent



ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent

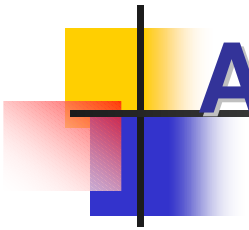


ARM architecture versions

- ARMv2a
  - ARM3 chip with on chip cache
  - included atomic load and store
  - coprocessor 15 : cache management
- ARMv3
  - 32-bit addressing, separate CPSR, SPSR
  - virtual memory support
  - ARM6, first processor after being independent

ARM architecture versions

- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution

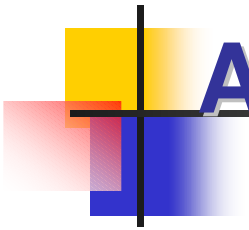


ARM architecture versions

- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution

ARM architecture versions

- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution

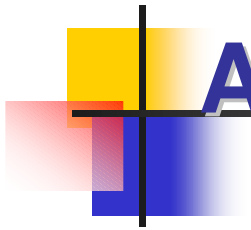


ARM architecture versions

- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution

ARM architecture versions

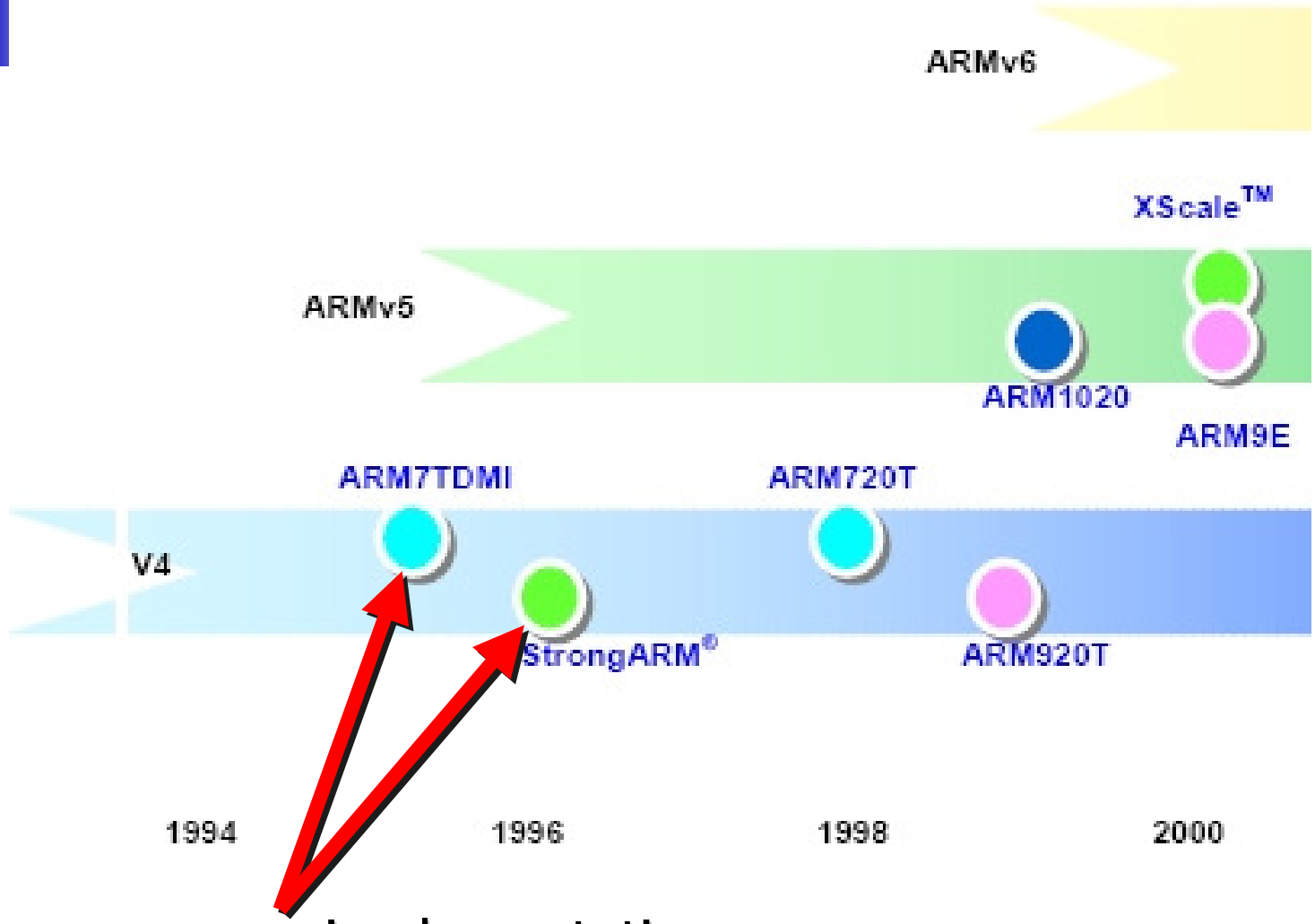
- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution



ARM architecture versions

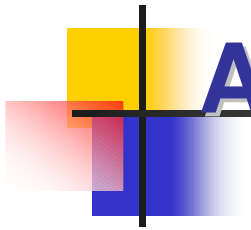
- ARMv4
  - added half word load and store
- ARMv5
  - improved ARM and Thumb interworking, count leading zeroes (CLZ) instruction
  - E: enhanced DSP instructions including saturated arithmetic and 16-bit multiply
  - J: support for accelerated Java bytecode execution

ARM architecture versions



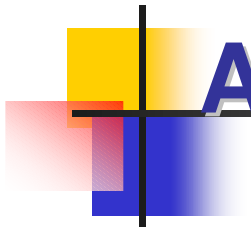
ARM architecture versions

- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A



ARM architecture versions

- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A

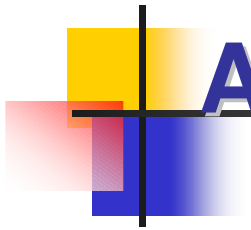


ARM architecture versions

- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A

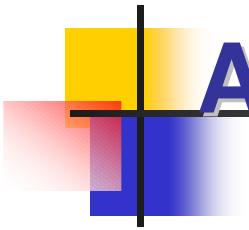
ARM architecture versions

- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A



ARM architecture versions

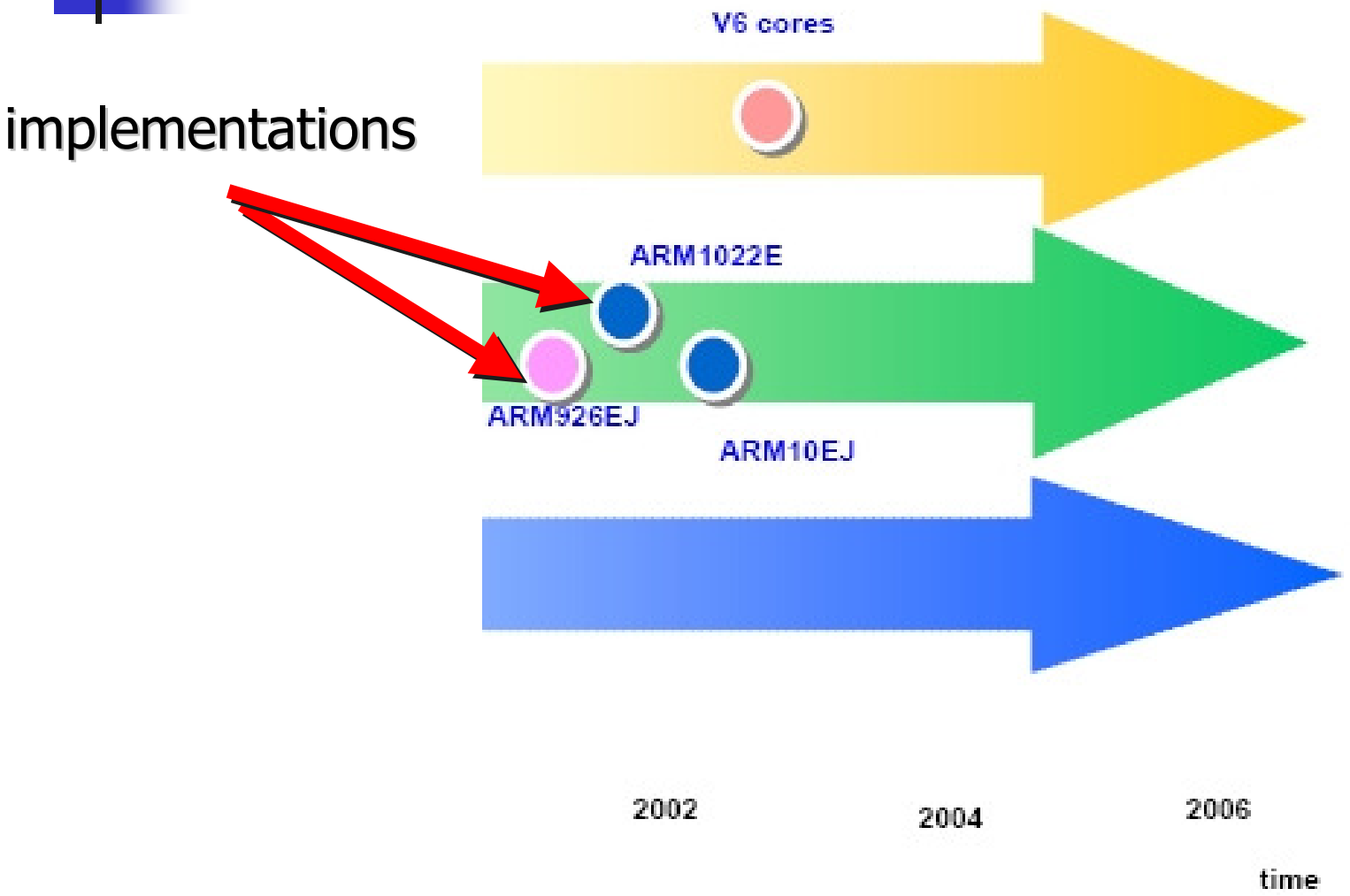
- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A



ARM architecture versions

- ARMv6
  - include TEJ enhancements
  - memory management
  - multiprocessing
  - SIMD instructions (media)
  - 6 new status bits: GE[3:0], E, A

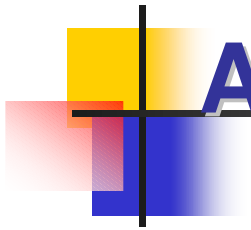
ARM architecture versions



ARM ISA features

Based on Berkeley RISC design

- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

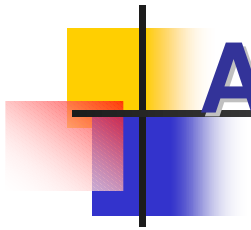
Based on Berkeley RISC design

- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions

ARM ISA features

Based on Berkeley RISC design

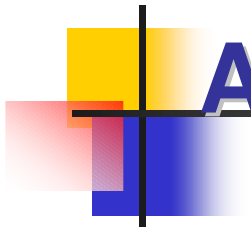
- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

Based on Berkeley RISC design

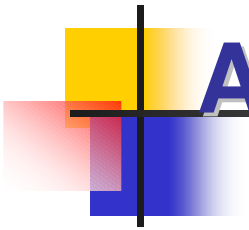
- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

Based on Berkeley RISC design

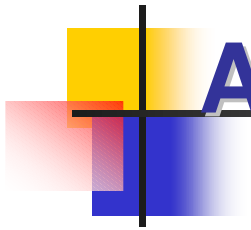
- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

Based on Berkeley RISC design

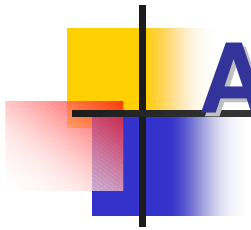
- Features used:
  - load-store architectures
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

Based on Berkeley RISC design

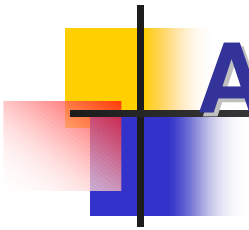
- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions



ARM ISA features

Based on Berkeley RISC design

- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions

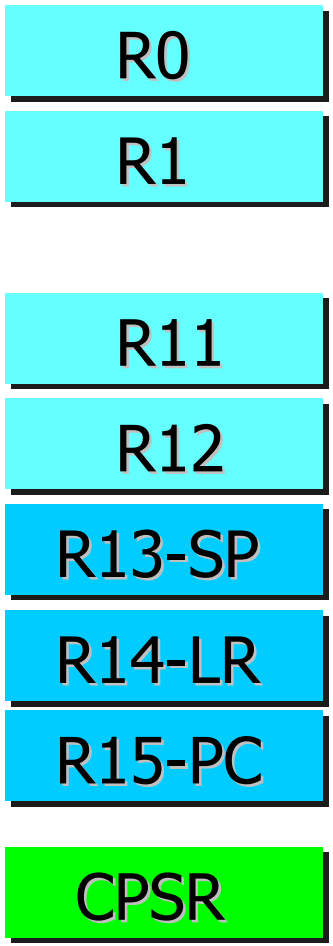


ARM ISA features

Based on Berkeley RISC design

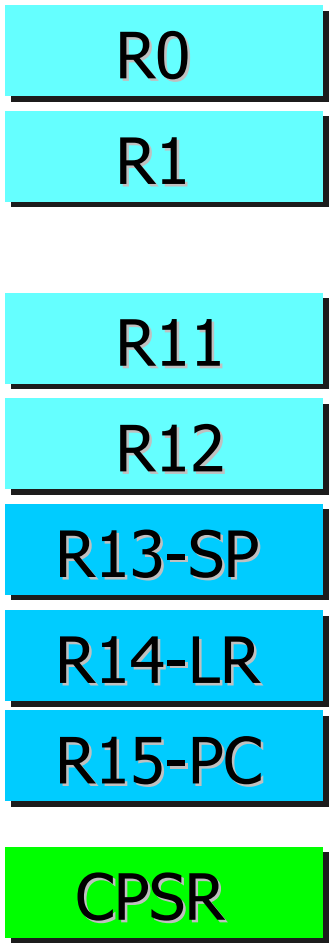
- Features used:
  - load-store architecture
  - fixed-length 32-bit instructions
  - 3-address instruction formats
- Features rejected
  - register windows
  - delayed branches
  - single-cycle execution of all instructions

ARM programming model



- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM programming model

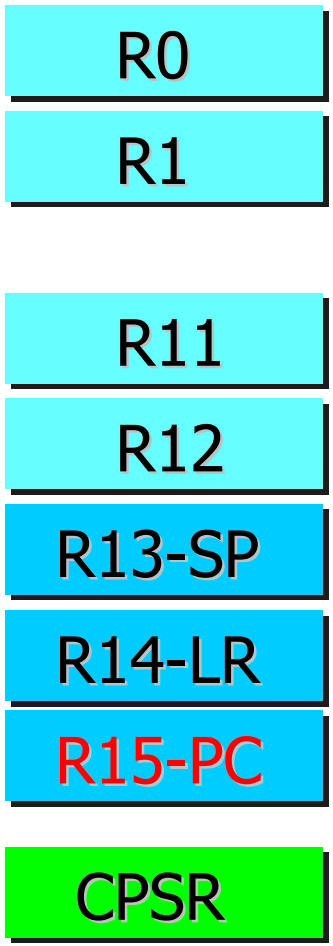


- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM programming model

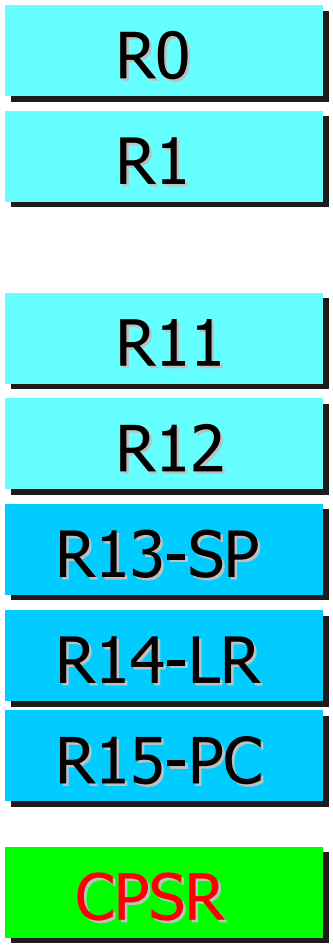
- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM programming model



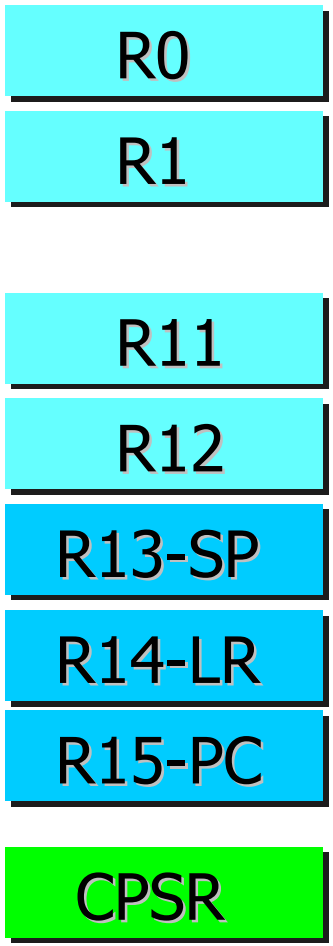
- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM programming model



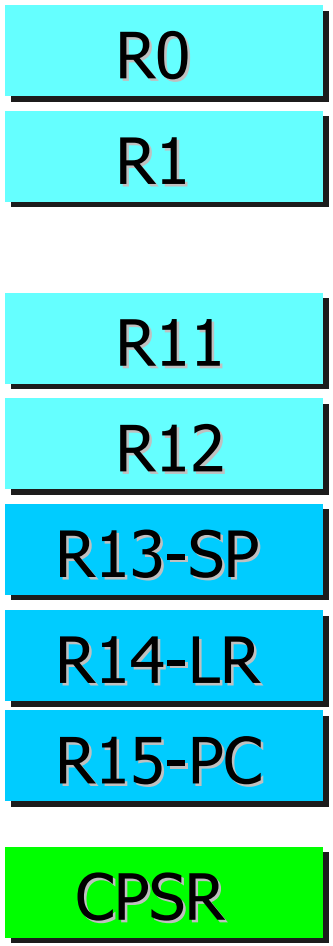
- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM programming model



- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

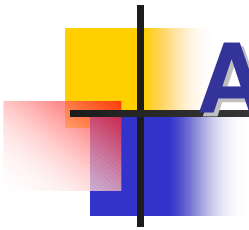
ARM programming model



- 32-bit instructions
- 17 visible registers
  - 15 general purpose
  - PC
  - CPSR
- 8/16/32 bits data types
- 7 modes of operation: usr, fiq, irq, svc, abt, sys, und

ARM coprocessor interface

- Supports a general-purpose extension of its instruction set through the addition of hardware coprocessors
- Support for up to 16 logical coprocessors
  - 16 private registers of any width
  - Coprocessors use load-store architecture
  - Coprocessors use handshaking to perform instructions

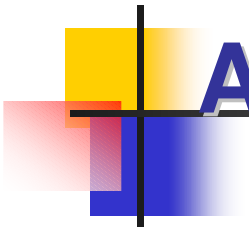


ARM coprocessor interface

- Supports a general-purpose extension of its instruction set through the addition of hardware coprocessors
- Support for up to 16 logical coprocessors
  - 16 private registers of any width
  - Coprocessors use load-store architecture
  - Coprocessors use handshaking to perform instructions

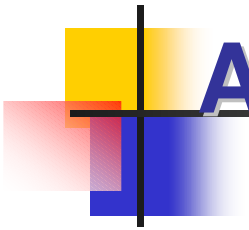
ARM coprocessor interface

- Supports a general-purpose extension of its instruction set through the addition of hardware coprocessors
- Support for up to 16 logical coprocessors
  - 16 private registers of any width
  - Coprocessors use load-store architecture
  - Coprocessors use handshaking to perform instructions



ARM coprocessor interface

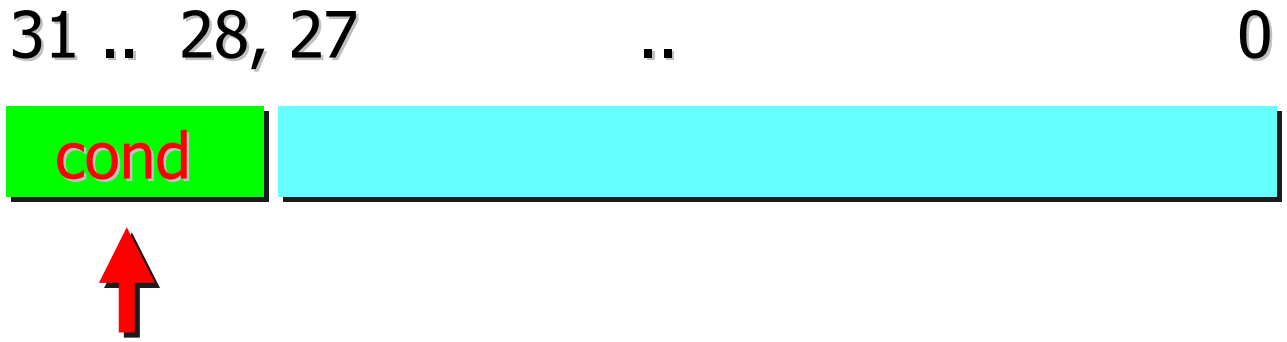
- Supports a general-purpose extension of its instruction set through the addition of hardware coprocessors
- Support for up to 16 logical coprocessors
  - 16 private registers of any width
  - Coprocessors use load-store architecture
  - Coprocessors use handshaking to perform instructions



ARM coprocessor interface

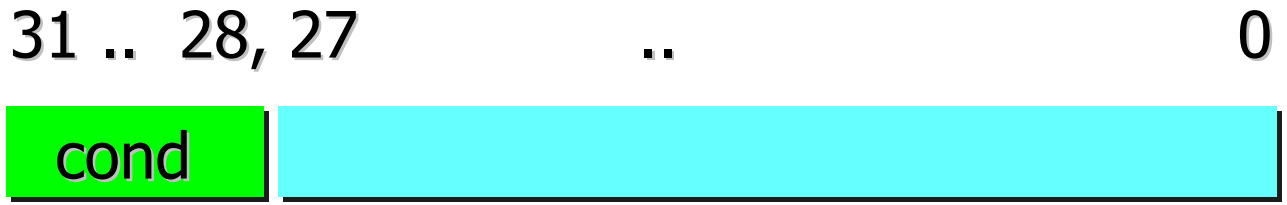
- Supports a general-purpose extension of its instruction set through the addition of hardware coprocessors
- Support for up to 16 logical coprocessors
  - 16 private registers of any width
  - Coprocessors use load-store architecture
  - Coprocessors use handshaking to perform instructions

Conditional execution



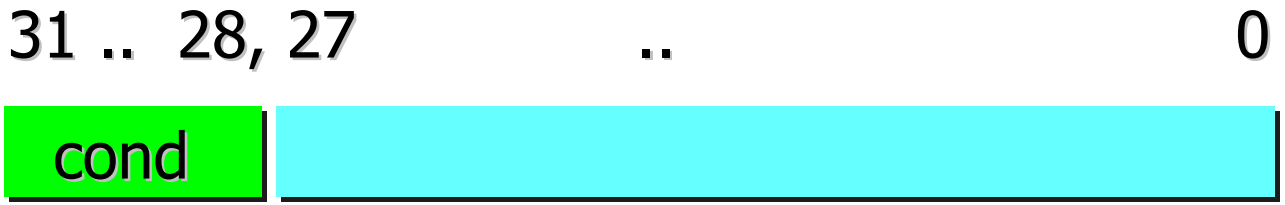
- Every instruction has a 4-bit condition code
- All instructions may be executed conditionally including:
  - supervisor instructions and
  - coprocessor instructions,
  - excluding Thumb instructions

Conditional execution



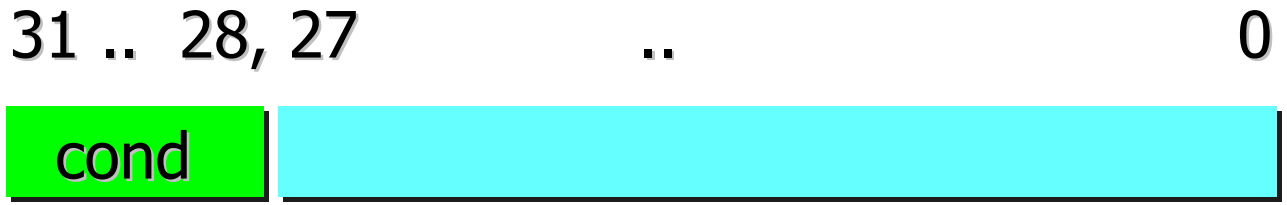
- Every instruction has a 4-bit condition code
- All instructions may be executed conditionally including:
  - supervisor instructions and
  - coprocessor instructions,
  - excluding Thumb instructions

Conditional execution



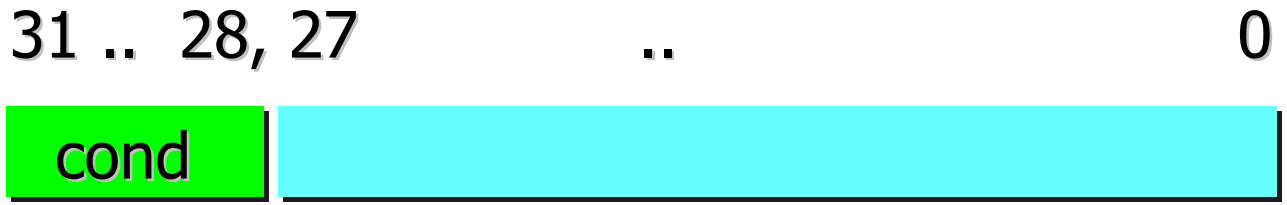
- Every instruction has a 4-bit condition code
- All instructions may be executed conditionally including:
  - supervisor instructions and
  - coprocessor instructions,
  - excluding Thumb instructions

Conditional execution



- Every instruction has a 4-bit condition code
- All instructions may be executed conditionally including:
  - supervisor instructions and
  - coprocessor instructions,
  - excluding Thumb instructions

Conditional execution



- Every instruction has a 4-bit condition code
- All instructions may be executed conditionally including:
  - supervisor instructions and
  - coprocessor instructions,
  - excluding Thumb instructions (16-bit instructions)

Conditional execution

Each instruction mnemonic may be extended by appending two letters: EQ,NE,GE,LE,GT, ..

```
subgt  Ri, Ri, Rj
suble  Ri, Ri, Rj
bne  loop
```

Conditional execution

Each instruction mnemonic may be extended by appending two letters: EQ,NE,GE,LE,GT, ..

```
subgt  Ri, Ri, Rj
suble  Ri, Ri, Rj
bne    loop
```

3-address arithmetical instructions

Conditional execution

Each instruction mnemonic may be extended by appending two letters: EQ,NE,GE,LE,GT, ..

```
subgt  Ri, Ri, Rj
suble  Ri, Ri, Rj
bne    loop
```

conditional branch instruction

Conditional execution

- conditional execution cuts down on the space available for displacement memory access
- avoids branch instructions for simple if statements

Conditional execution

```
int gcd(int i, int j)
{
while (i!=j)
{ if (i>j) i -=j;
else j -=i; }
return i;
}
```

b	test	
loop	subgt	Ri, Ri, Rj
	suble	Rj, Rj, Ri
test	cmp	Ri, Rj
	bne	loop

Conditional execution

```
int gcd(int i, int j)
{
while (i!=j)
{ if (i>j) i -=j;
else j -=i; }
return i;
}
```

```
b       test
loop    subgt   Ri, Ri, Rj
        suble   Rj, Rj, Ri
test    cmp     Ri, Rj
        bne     loop
```

Conditional execution

```
int gcd(int i, int j)
{
while (i!=j)
{ if (i>j) i -=j;
else j -=i; }
return i;
}
```

b	test	
loop	subgt	Ri, Ri, Rj
	suble	Rj, Rj, Ri
test	cmp	Ri, Rj
	bne	loop

Multiple register transfer operation

- Any subset (or all) of the 16 registers visible in the current operating mode to be loaded from or stored to memory
- Used on procedure entry and return to save and restore workspace registers
- Useful for high-bandwidth memory block copy routines.

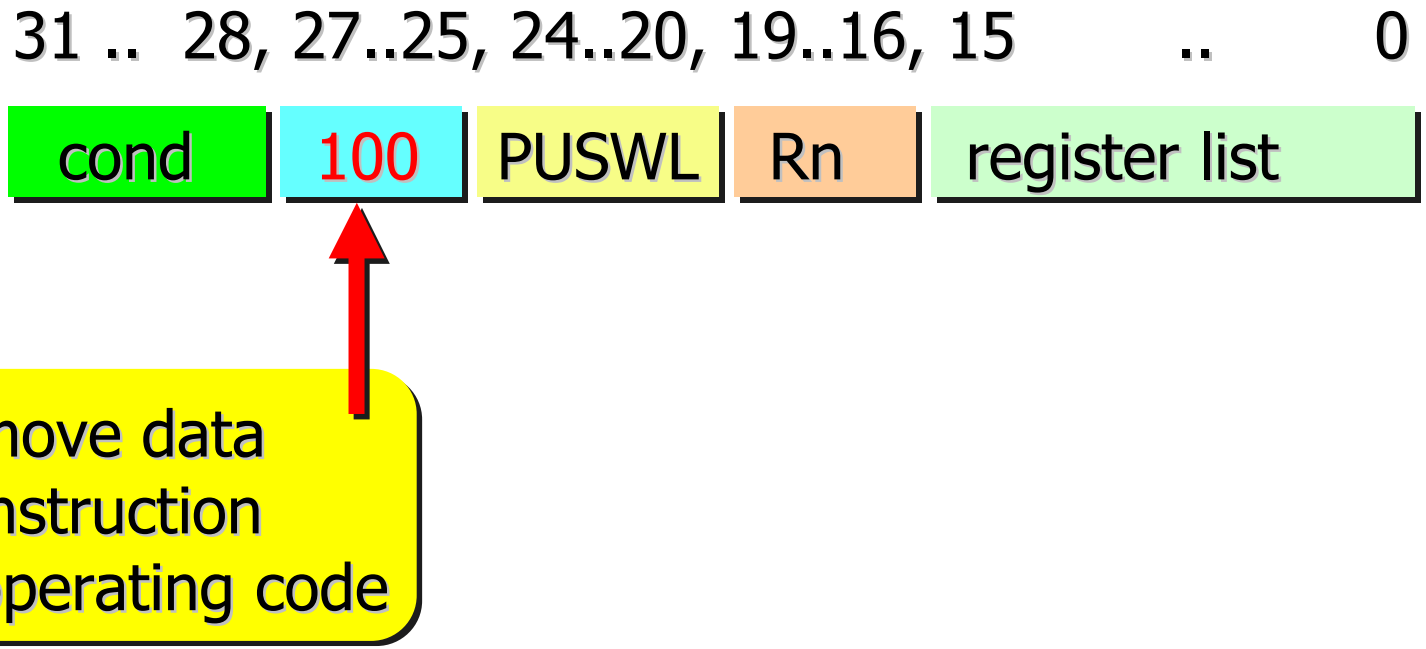
Multiple register transfer operation

- Any subset (or all) of the 16 registers visible in the current operating mode to be loaded from or stored to memory
- Used on procedure entry and return to save and restore workspace registers
- Useful for high-bandwidth memory block copy routines.

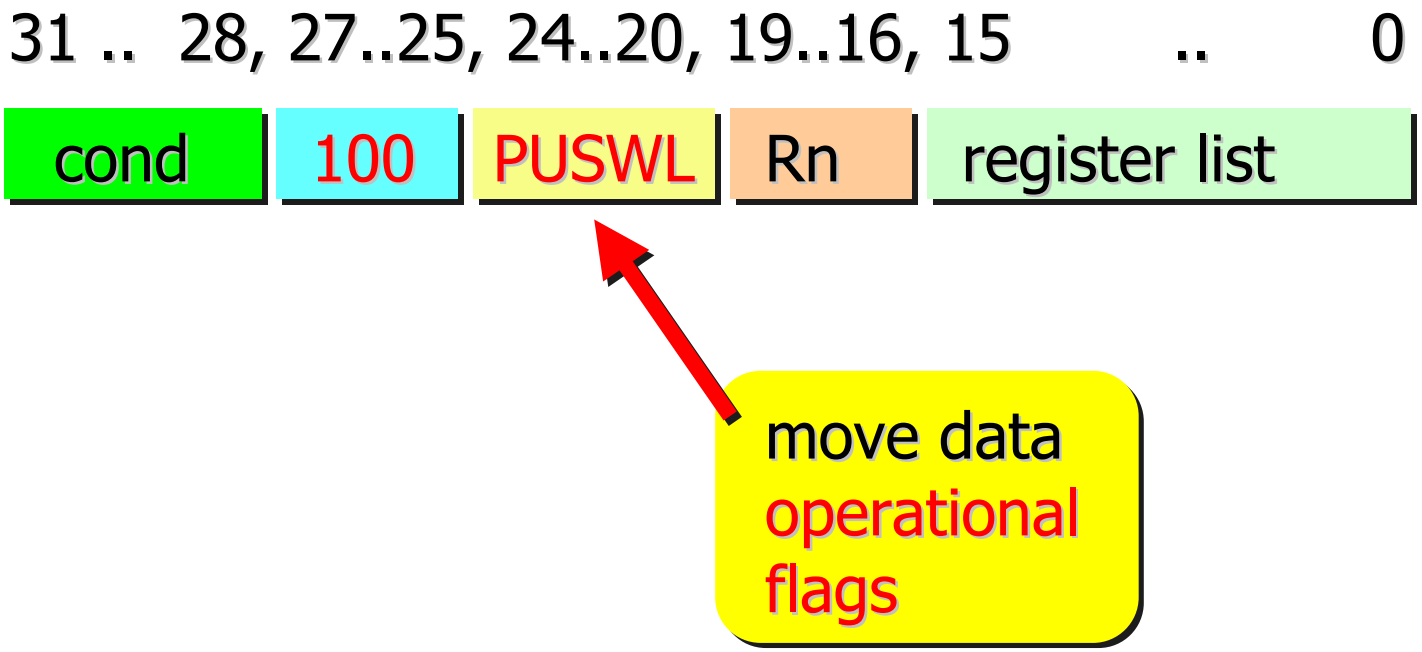
Multiple register transfer operation

- Any subset (or all) of the 16 registers visible in the current operating mode to be loaded from or stored to memory
- Used on procedure entry and return to save and restore workspace registers
- Useful for high-bandwidth memory block copy routines.

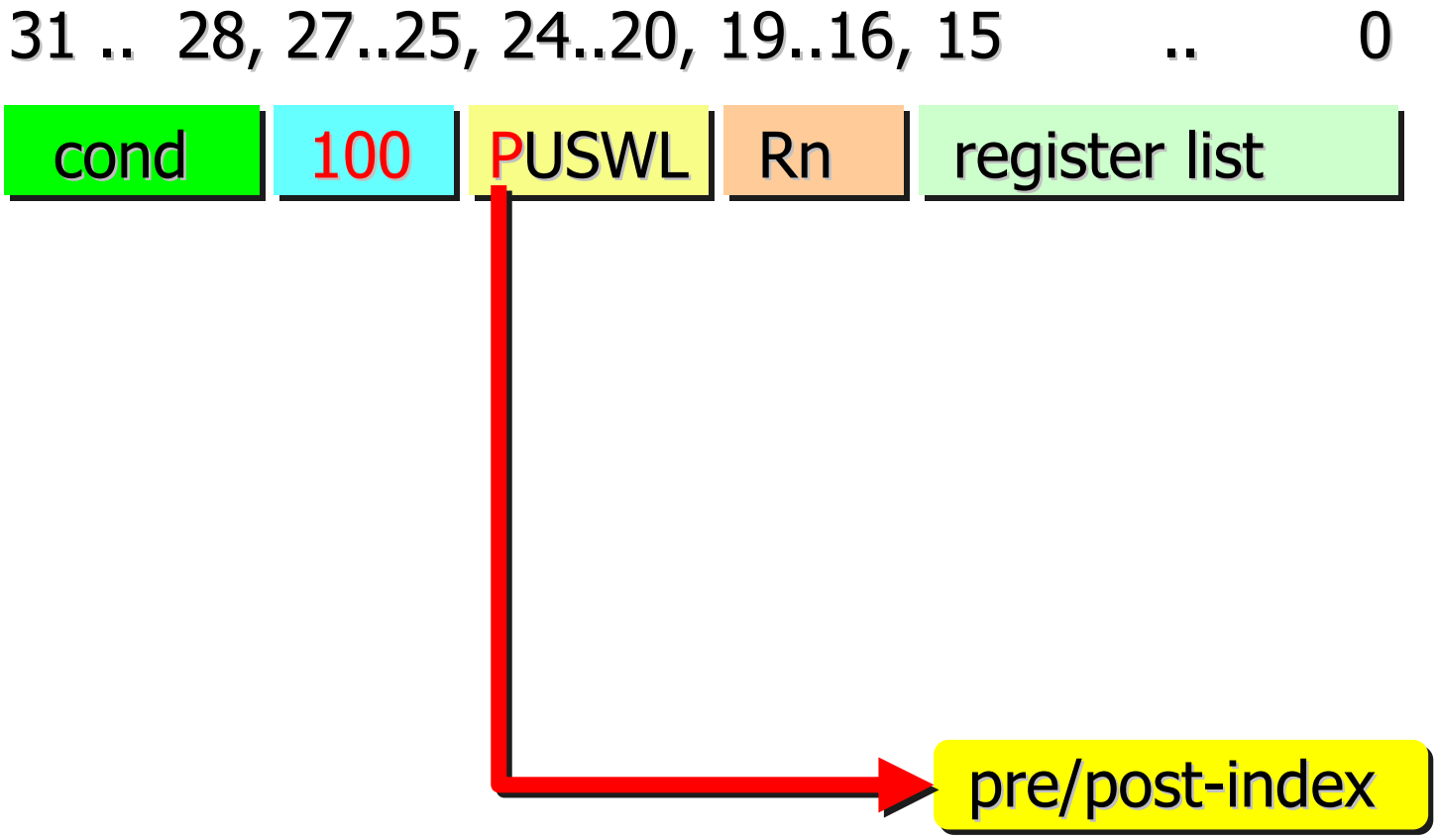
Multiple register transfer operation



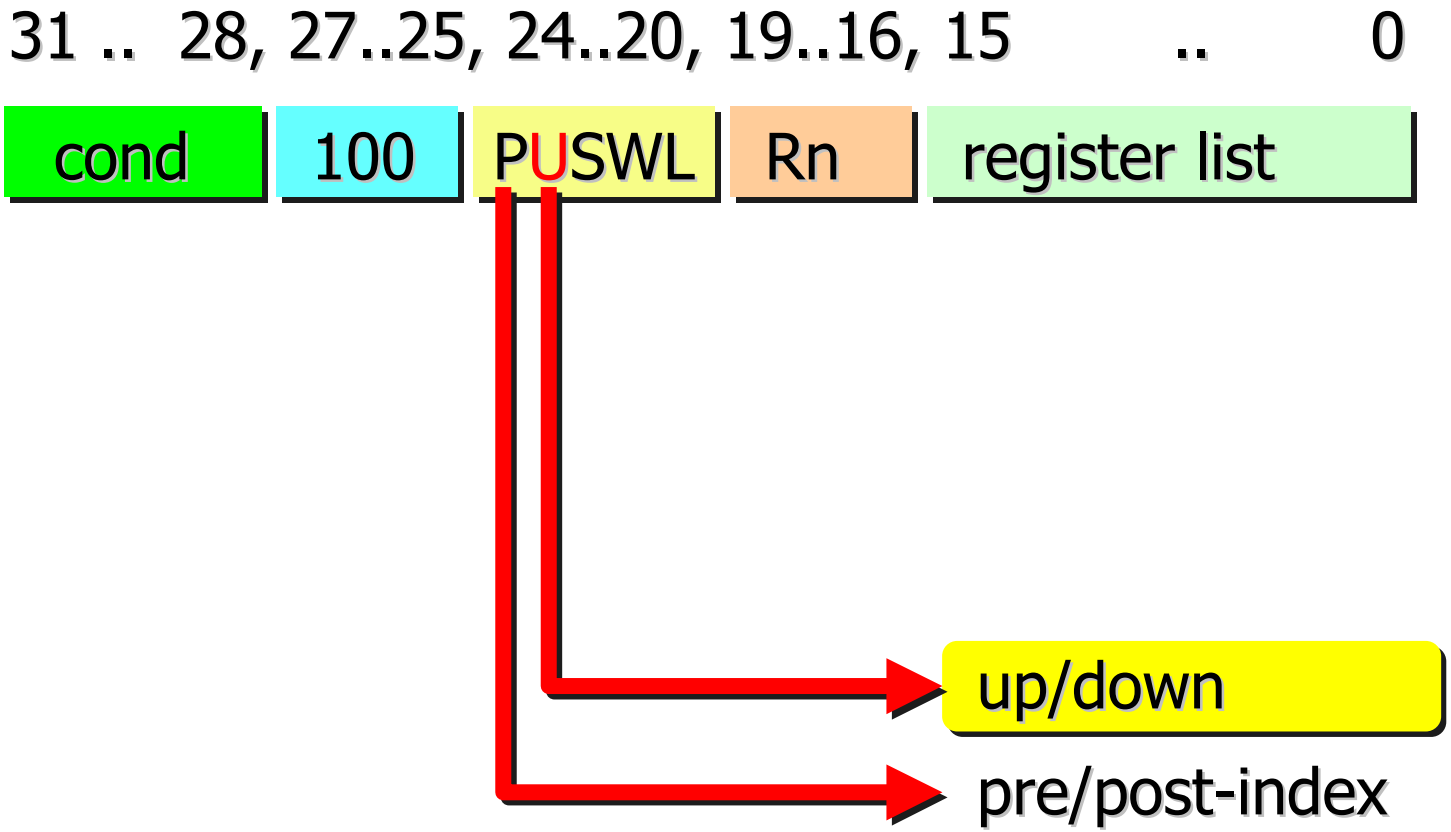
Multiple register transfer operation



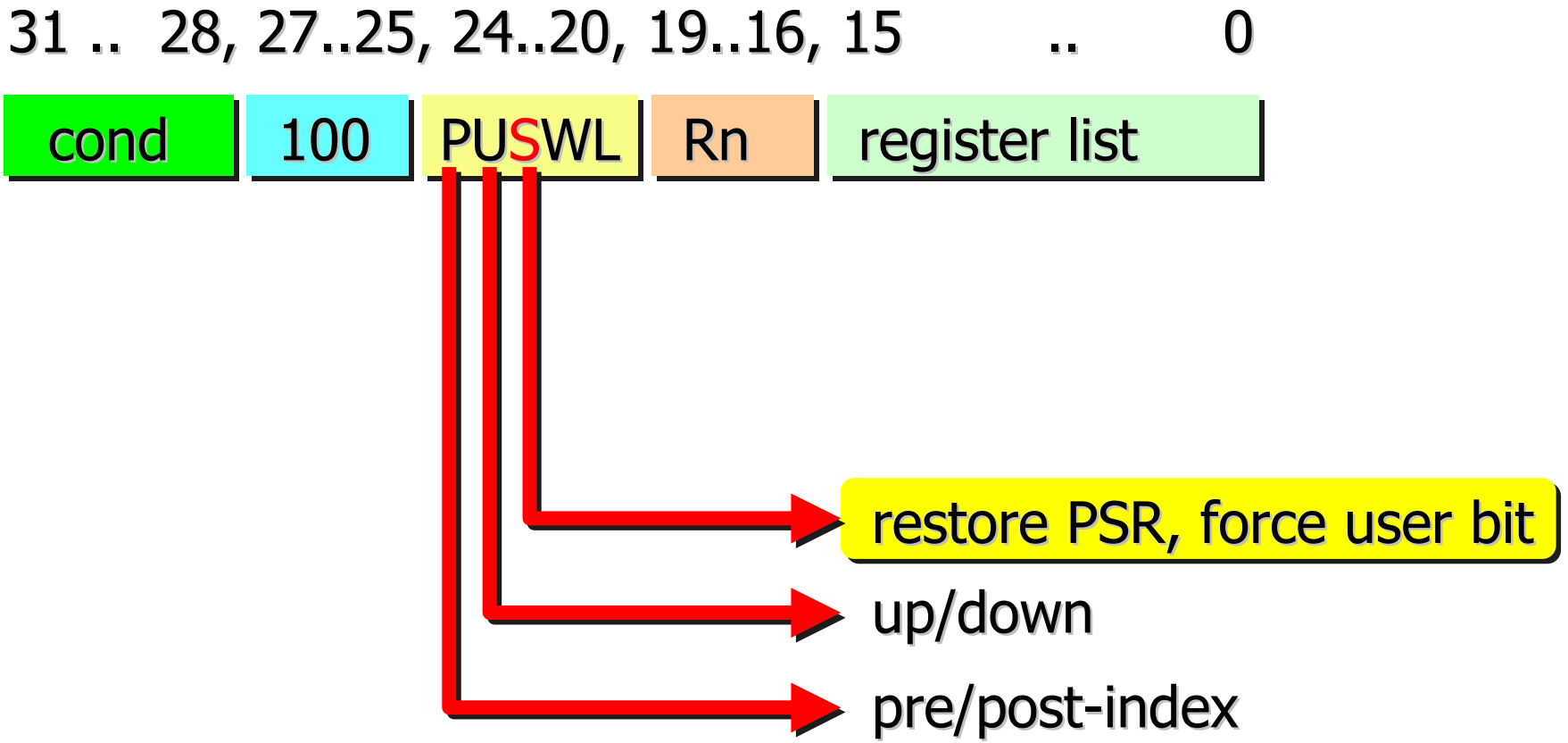
Multiple register transfer operation



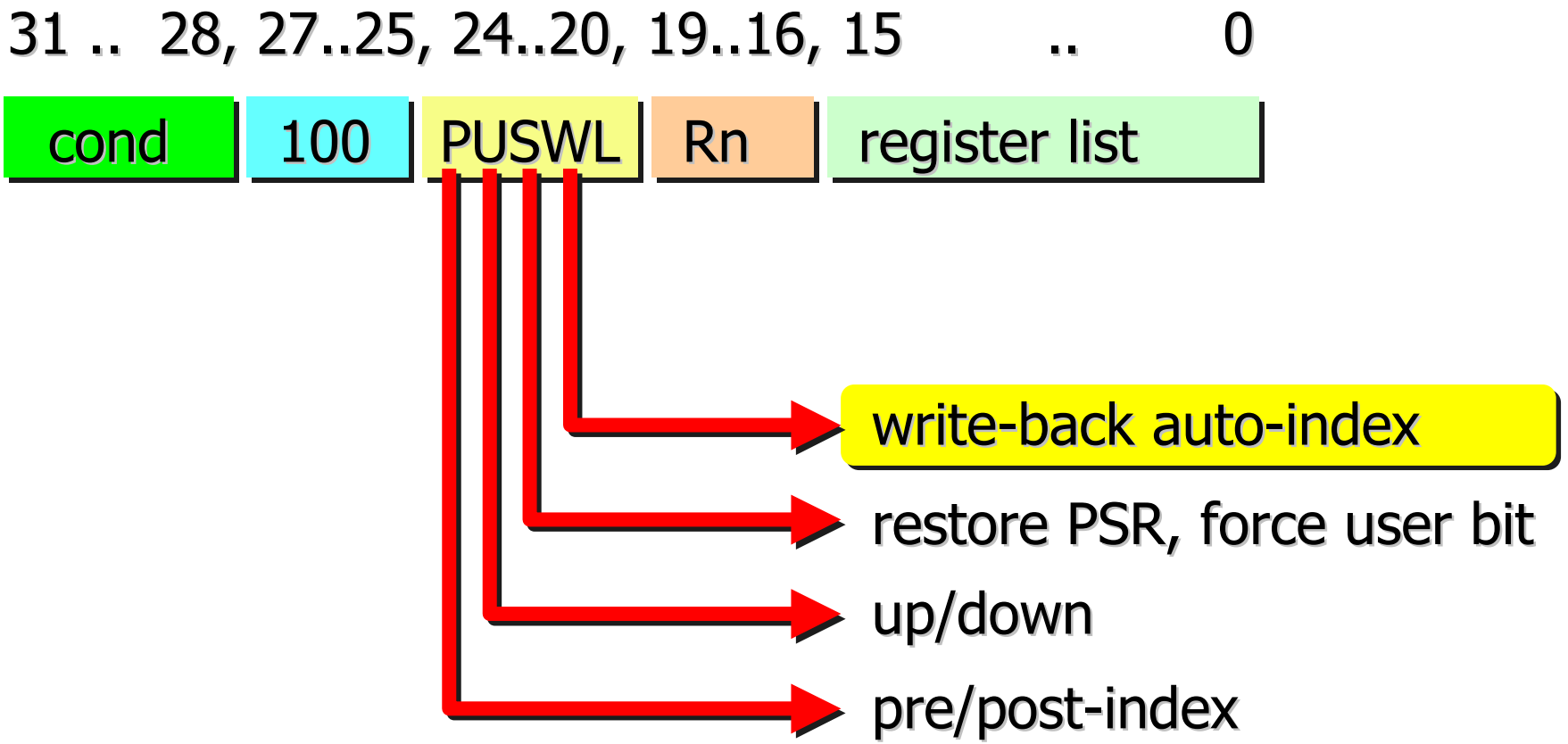
Multiple register transfer operation



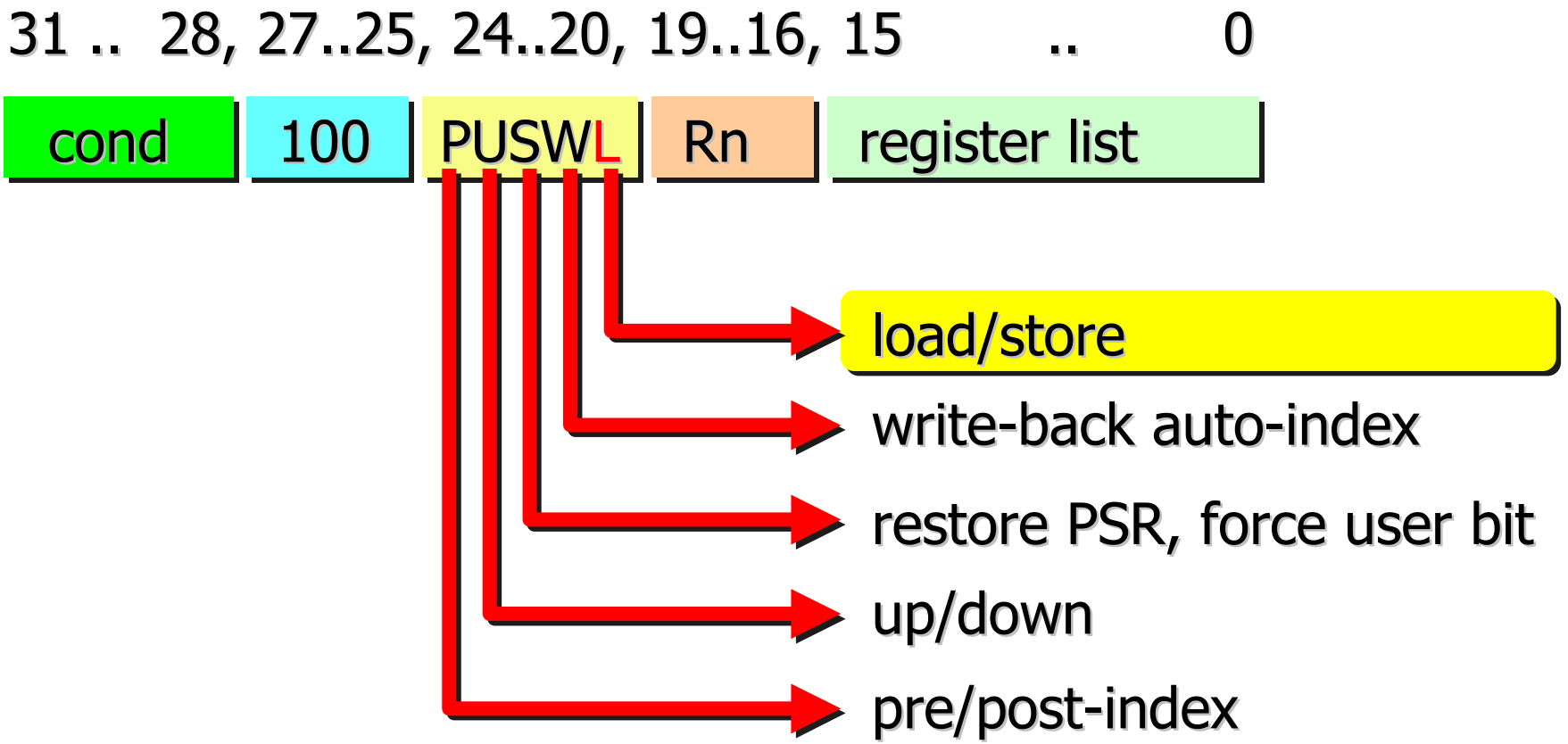
Multiple register transfer operation



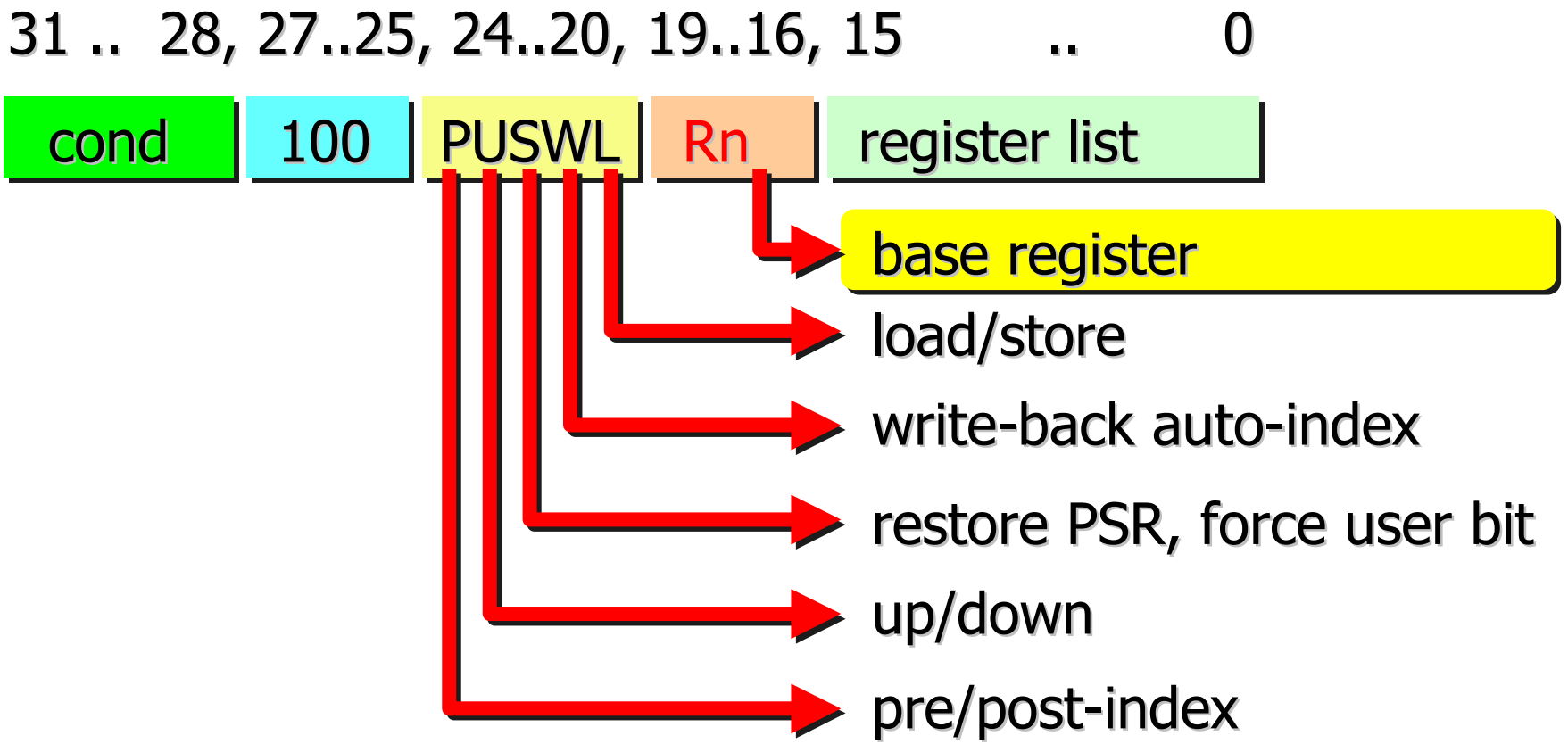
Multiple register transfer operation



Multiple register transfer operation



Multiple register transfer operation



Shifter and ALU operations

- The operand can be shifted before being processed by ALU and stored into a destination register

- Operations include arithmetic, logical, and register-register move

- i+=(j<<3) can be performed as a single instruction on the ARM

Shifter and ALU operations

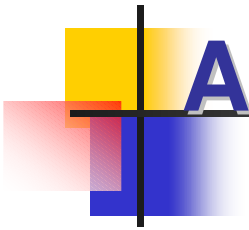
- The operand can be shifted before being processed by ALU and stored into a destination register
- Operations include arithmetic, logical, and register-register move
- i+=(j<<3) can be performed as a single instruction on the ARM

Shifter and ALU operations

- The operand can be shifted before being processed by ALU and stored into a destination register
- Operations include arithmetic, logical, and register-register move
- i+=(j<<3) can be performed as a single instruction on the ARM

Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

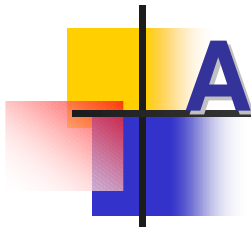


Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals



Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

Architectural support for OS

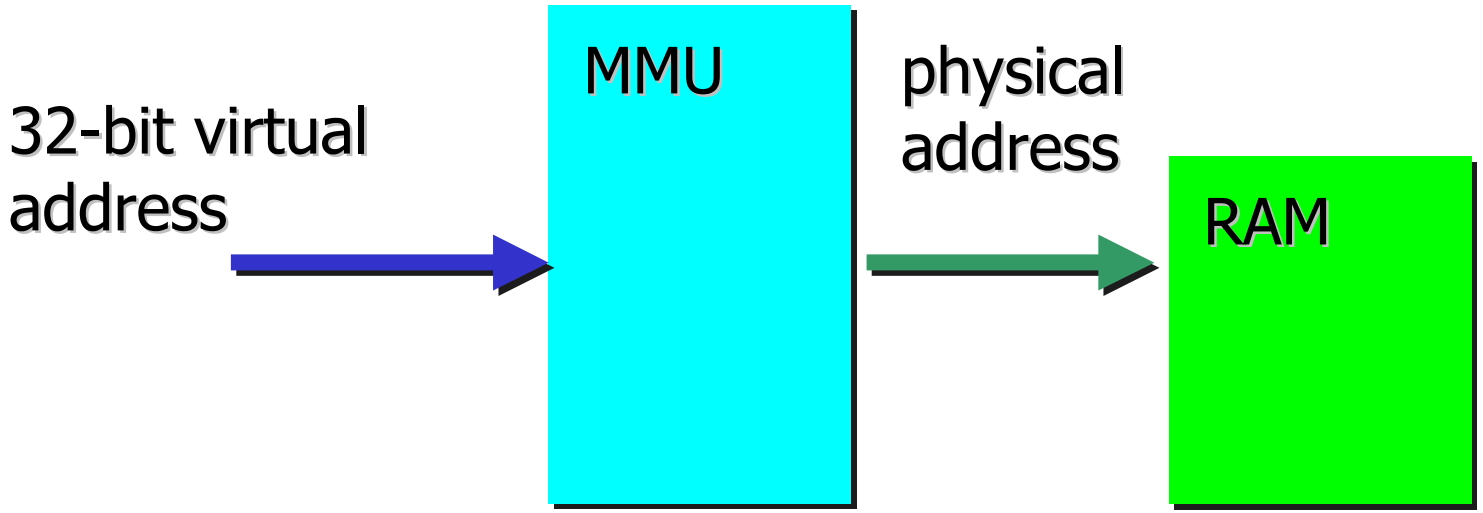
- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

Architectural support for OS

- coprocessor number 15
- on-chip system OS control processor controls:
  - cache memory
  - memory management and protection
  - pre-fetch buffer
  - branch target cache
  - system configuration signals

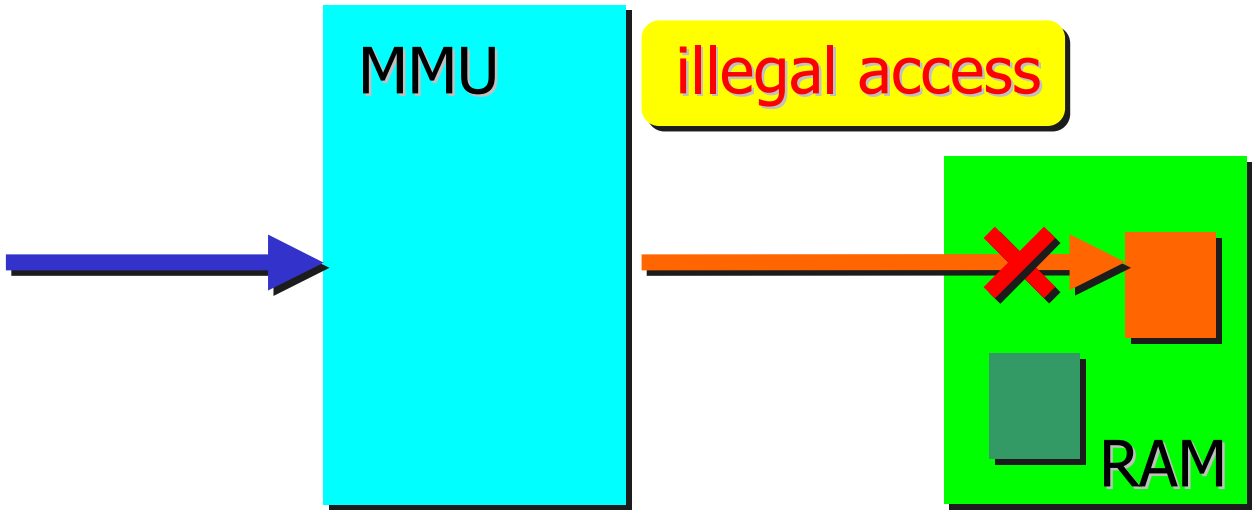
MMU architecture operations

- translates virtual addresses into physical addresses



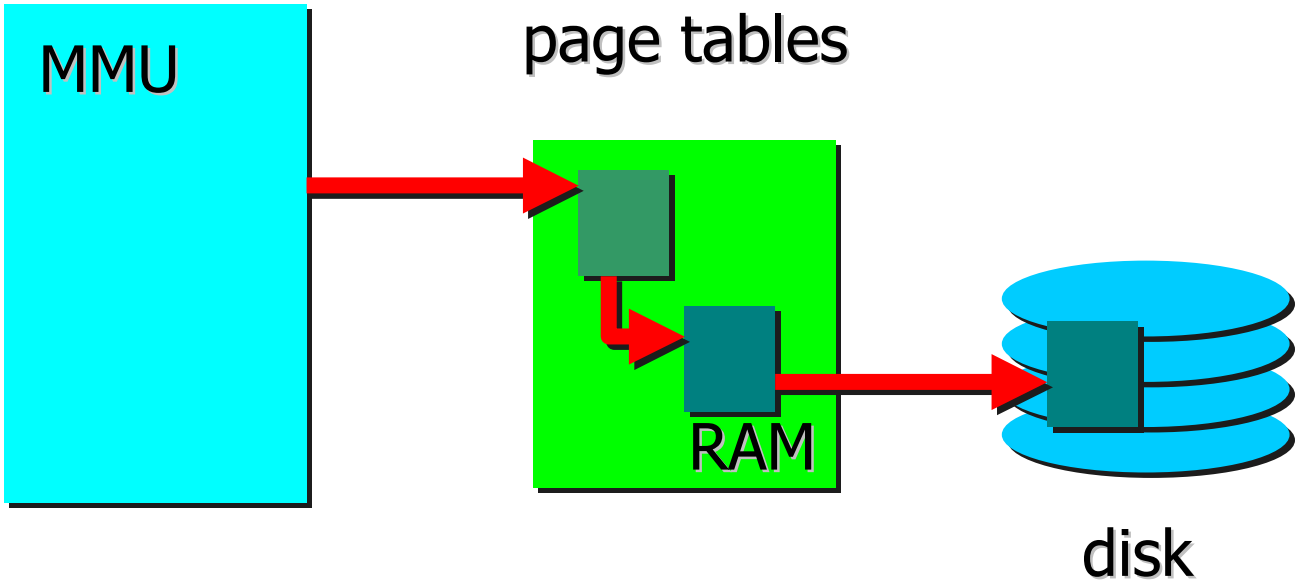
MMU architecture operations

- translates virtual addresses into physical addresses
- controls memory access permissions, aborting illegal accesses



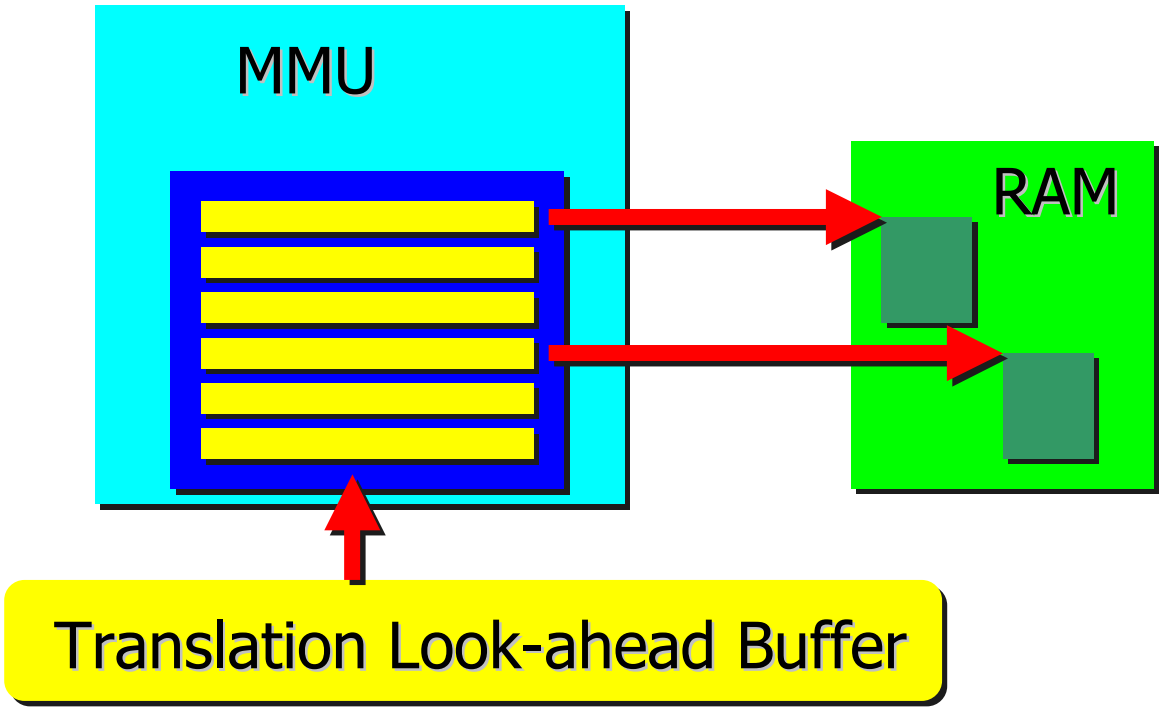
MMU architecture operations

- uses two-level page table with table-walking hardware



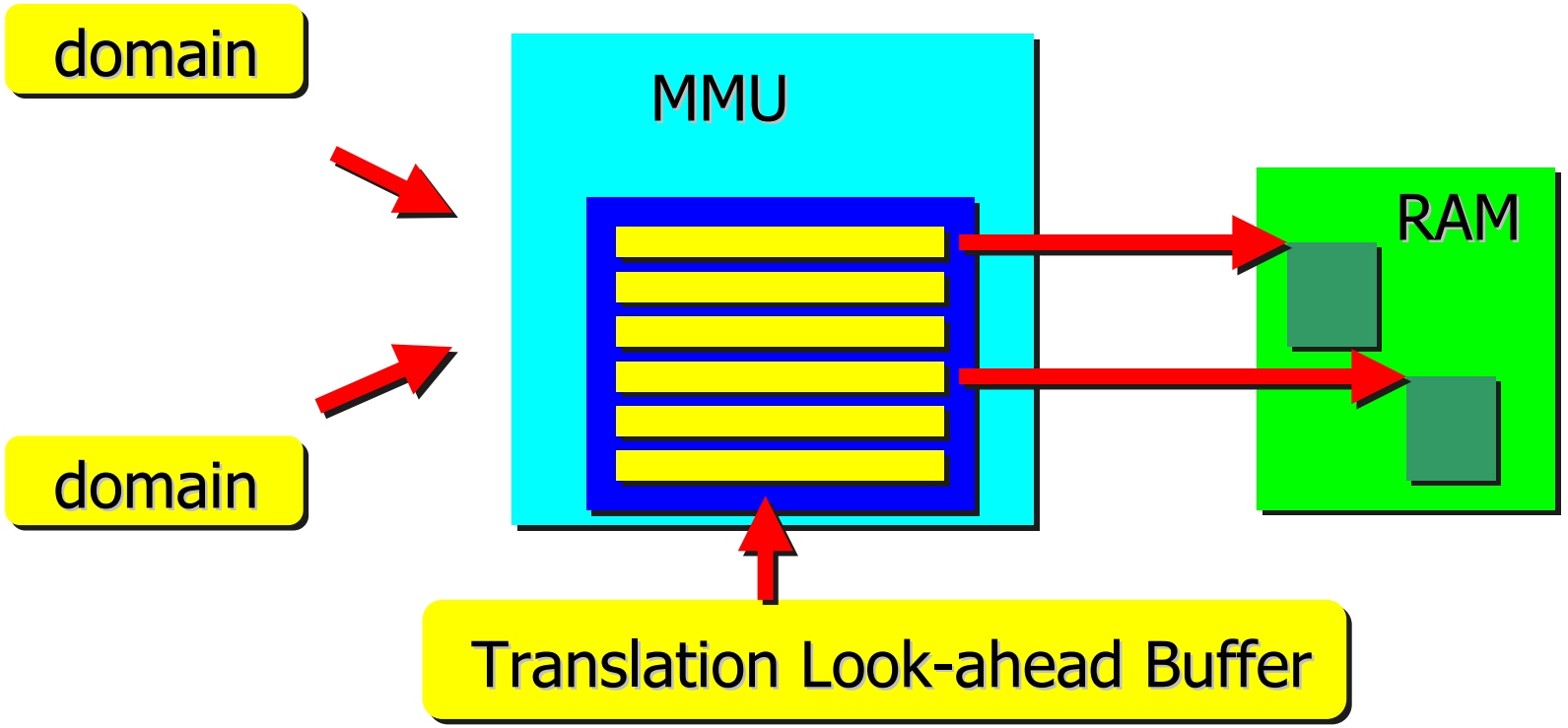
MMU architecture operations

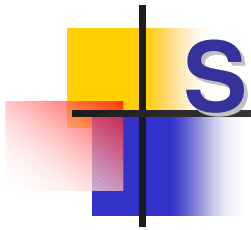
- controls a TLB which stories recently used page translations



MMU architecture operations

- provides 16 domains – each protected from one another while using the same TLB

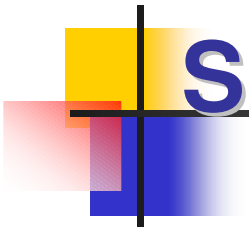




Support for OS

Synchronization problem:

- mutually exclusive access to data structure
- only one process can access this at any time
- must wait until no other process is accessing the data
- need of a lock mechanism to prevent another process to access the data



Support for OS

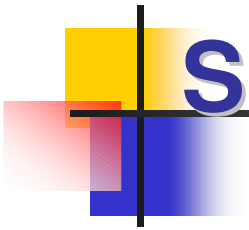
Synchronization problem:

- mutually exclusive access to data structure
- only one process can access this at any time
- must wait until no other process is accessing the data
- need of a lock mechanism to prevent another process to access the data

Support for OS

Synchronization problem:

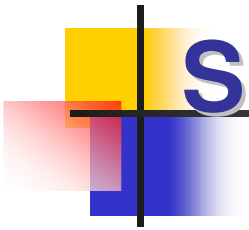
- mutually exclusive access to data structure
- only one process can access this at any time
- must wait until no other process is accessing the data
- need of a lock mechanism to prevent another process to access the data



Support for OS

Synchronization problem:

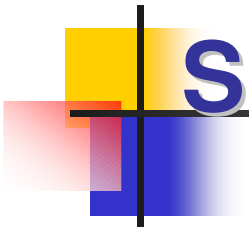
- mutually exclusive access to data structure
- only one process can access this at any time
- must wait until no other process is accessing the data
- need of a lock mechanism to prevent another process to access the data



Support for OS

Synchronization problem:

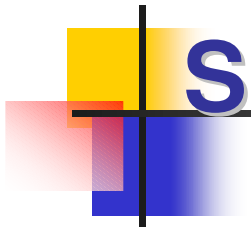
- mutually exclusive access to data structure
- only one process can access this at any time
- must wait until no other process is accessing the data
- need of a lock mechanism to prevent another process to access the data



Support for OS

ARM architecture supports the synchronization by providing a "swap" instruction:

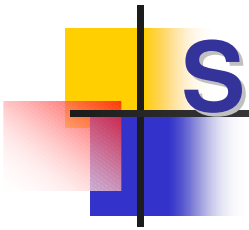
- "swap" instruction is atomic
- performs test and set operation
- a register is set to the "busy" value and swapped with the memory location containing the Boolean
- if loaded value is interpreted as "free" the process may continue the execution; otherwise must wait by on the lock



Support for OS

ARM architecture supports the synchronization by providing a "swap" instruction:

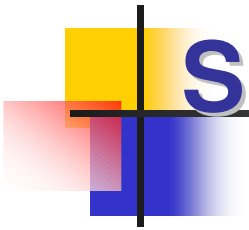
- "swap" instruction is atomic
- performs test and set operation
- a register is set to the "busy" value and swapped with the memory location containing the Boolean
- if loaded value is interpreted as "free" the process may continue the execution; otherwise must wait by on the lock



Support for OS

ARM architecture supports the synchronization by providing a "swap" instruction:

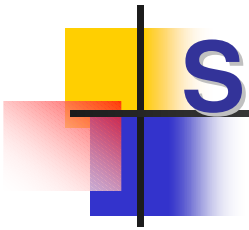
- "swap" instruction is atomic
- performs test and set operation
- a register is set to the "busy" value and swapped with the memory location containing the Boolean
- if loaded value is interpreted as "free" the process may continue the execution; otherwise must wait by on the lock



Support for OS

ARM architecture supports the synchronization by providing a "swap" instruction:

- "swap" instruction is atomic
- performs test and set operation
- a register is set to the "busy" value and swapped with the memory location containing the Boolean
- if loaded value is interpreted as "free" the process may continue the execution; otherwise must wait by on the lock



Support for OS

ARM architecture supports the synchronization by providing a "swap" instruction:

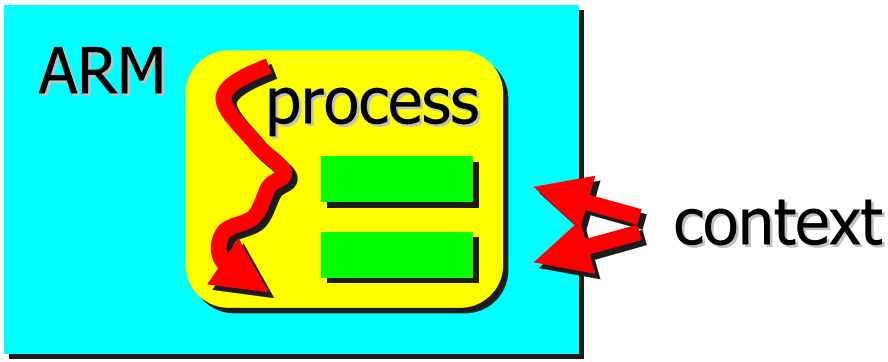
- "swap" instruction is atomic
- performs test and set operation
- a register is set to the "busy" value and swapped with the memory location containing the Boolean
- if loaded value is interpreted as "free" the process may continue the execution; otherwise must wait by on the lock

Context switching

- a process runs in a context

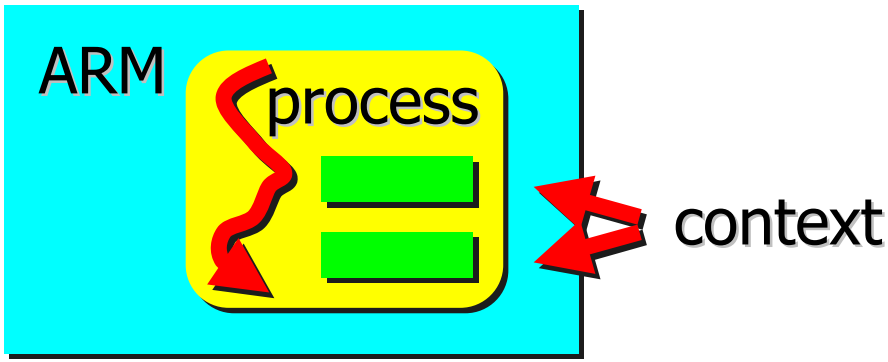
- context state includes the values of all registers including the program counter, stack pointer, etc.

- when a process switch takes place the context of the old process must be saved and that of the new process must be restored



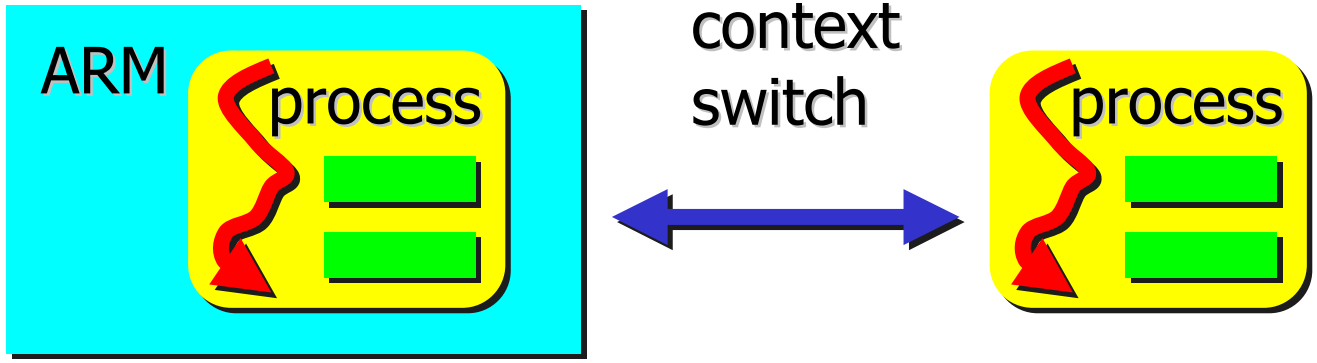
Context switching

- a process runs in a context
- context state includes the values of all registers including the program counter, stack pointer, etc.
- when a process switch takes place the context of the old process must be saved and that of the new process must be restored



Context switching

- a process runs in a context
- context state includes the values of all registers including the program counter, stack pointer, etc.
- when a process switch takes place the context of the old process must be saved and that of the new process must be restored



Context switching

- ARM provides architectural support for register saving and restoring in privileged mode
- special forms of the load and store multiple instructions
- allows code running in a non-user mode to save and restore the user registers from an area of memory addressed by a non-user mode register

Context switching

- ARM provides architectural support for register saving and restoring in privileged mode
- special forms of the load and store multiple instructions
- allows code running in a non-user mode to save and restore the user registers from an area of memory addressed by a non-user mode register

Context switching

- ARM provides architectural support for register saving and restoring in privileged mode
- special forms of the load and store multiple instructions
- allows code running in a non-user mode to save and restore the user registers from an area of memory addressed by a non-user mode register

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

AMBA interface

- de-facto standard for on-chip bus
- open standard
- framework for System-on-Chip designs
- strategy for interconnection and management of functional blocks (SoC)
- one or more CPUs with multiple peripherals
- maximum confidence in peripheral reuse
- IP designers develop own products without worrying about connectivity

ARM versions – an overview

Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

ARM versions – an overview

Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

ARM versions – an overview

Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

ARM versions – an overview

Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

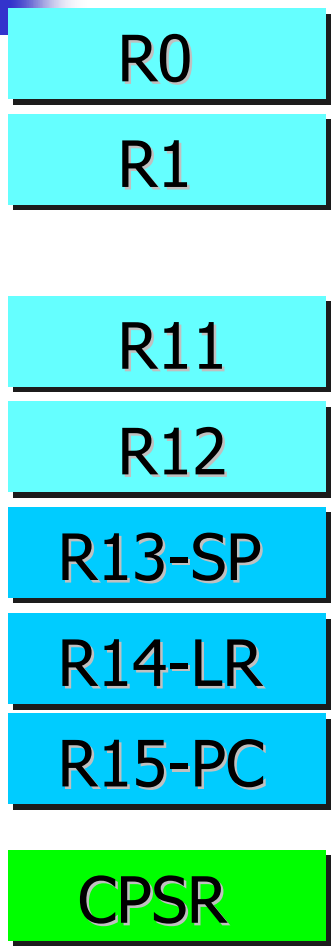
ARM versions – an overview

Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

ARM versions – an overview

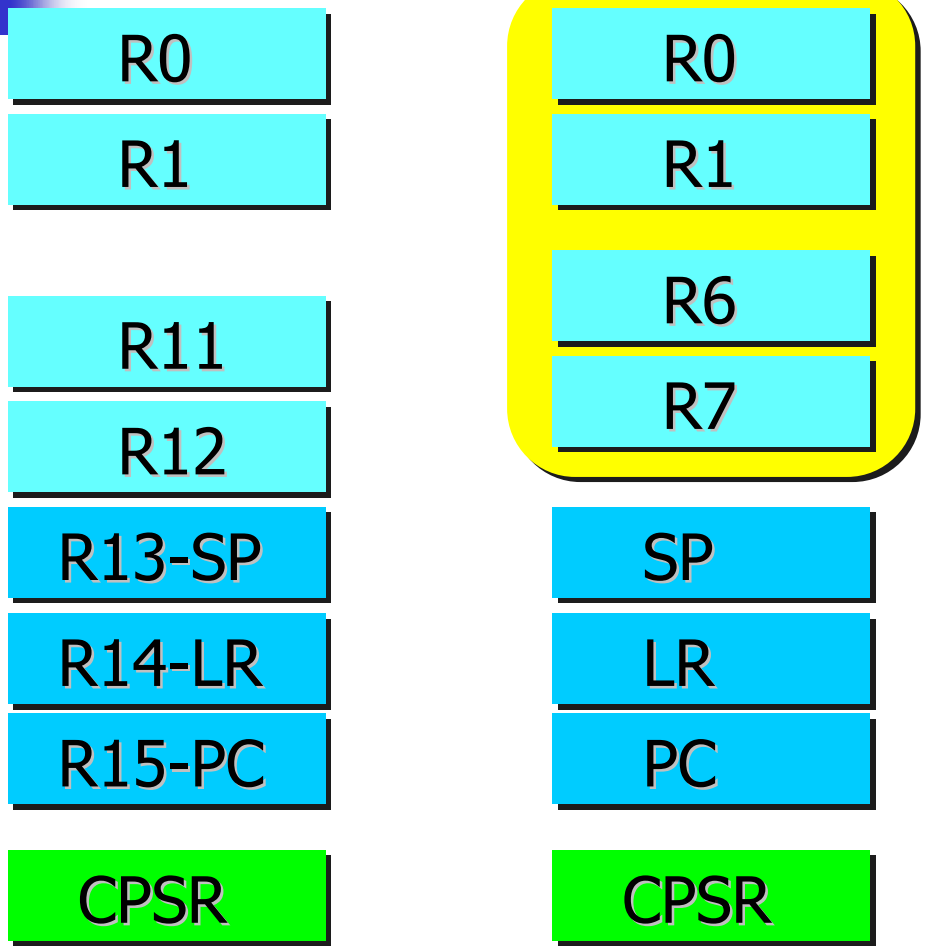
Version	Thumb	DSP	Jazelle	Media	TrustZone	Thumb2
v4T	yes					
v5TE	yes	yes				
v5TEJ	yes	yes	yes			
v6	yes	yes	yes	yes		
v6Z	yes	yes	yes	yes	yes	
v6T2	yes	yes	yes	yes		yes

Thumb Instruction Set



16-bit instructions mapped to 32-bit ARM instructions

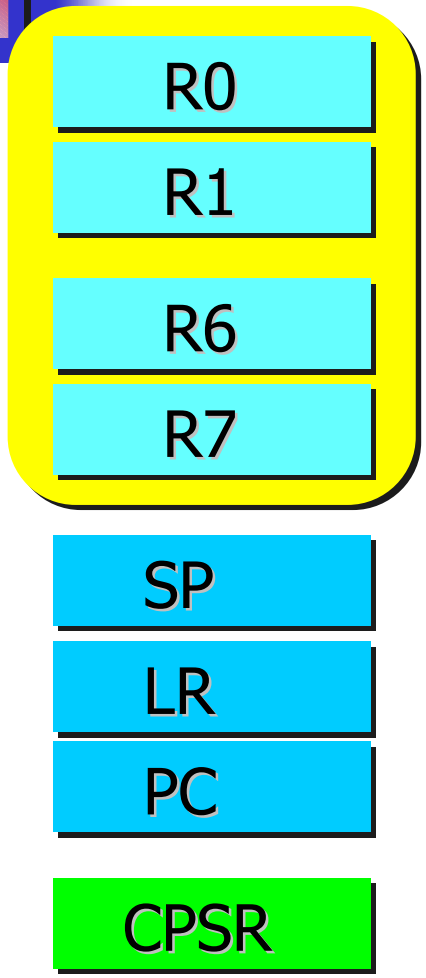
Thumb Instruction Set



only 8 visible registers

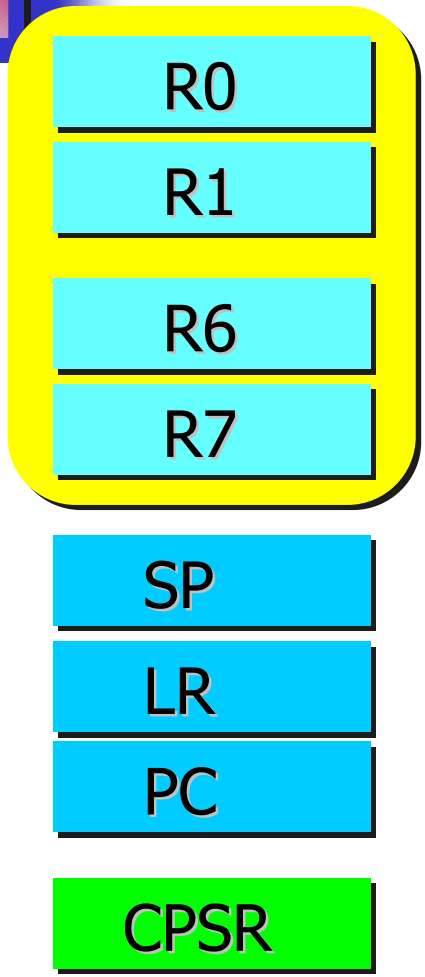
16-bit instructions mapped to 32-bit ARM instructions

Thumb Instruction Set



- CPSR – (Current Program Status Register) determines the mode of operation
- switching the mode by the execution of Branch and Exchange instruction

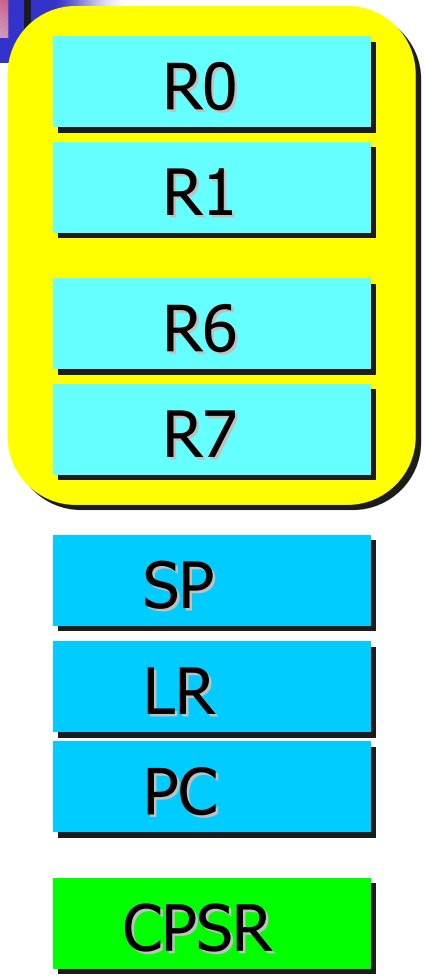
Thumb Instruction Set



- CPSR – (Current Program Status Register) determines the mode of operation

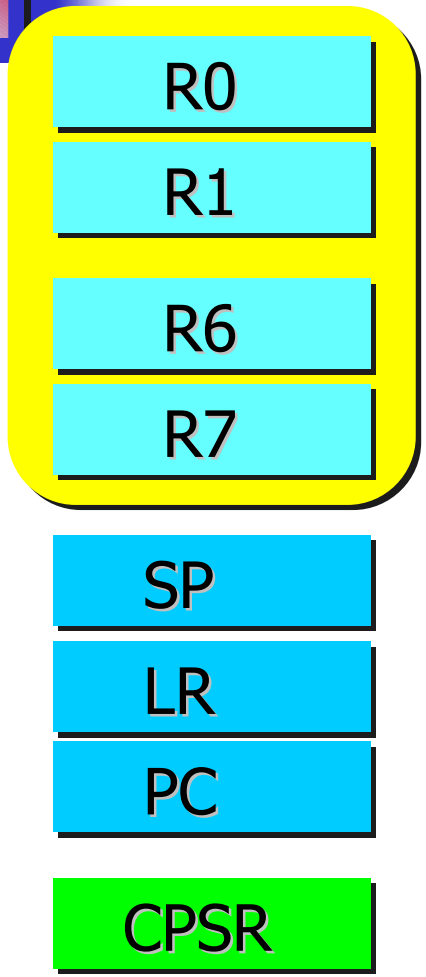
- switching the mode by the execution of Branch and Exchange instruction

Thumb – ARM differences



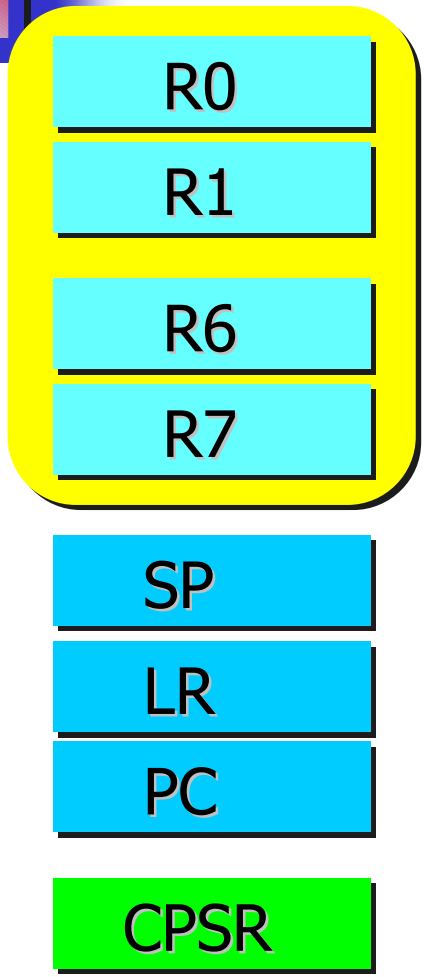
- most Thumb instructions are executed unconditionally
- data processing instructions use two-address format
- instruction formats are less regular than ARM instruction formats

Thumb – ARM differences

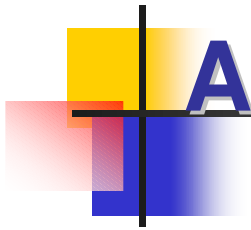


- most Thumb instructions are executed unconditionally
- data processing instructions use two-address format
- instruction formats are less regular than ARM instruction formats

Thumb – ARM differences

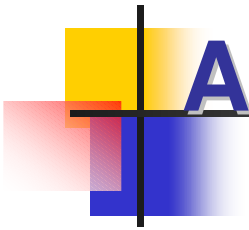


- most Thumb instructions are executed unconditionally
- data processing instructions use two-address format
- instruction formats are less regular than ARM instruction formats



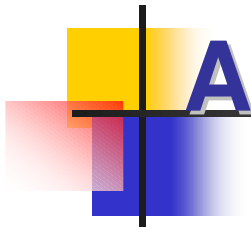
ARM version 5

- ARM DSP extensions:
  - broaden the suitability of the ARM CPU for intensive signal processing (audio, video)
- Jazelle
  - architectural extensions to execute Java Byte Code directly



ARM version 5

- ARM DSP extensions:
  - broaden the suitability of the ARM CPU for intensive signal processing (audio, video)
- Jazelle
  - architectural extensions to execute Java Byte Code directly

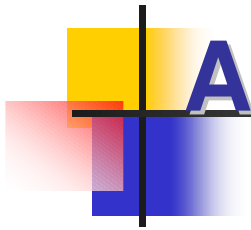


ARM version 5

- ARM DSP extensions:
  - broaden the suitability of the ARM CPU for intensive signal processing (audio, video)
- Jazelle
  - architectural extensions to execute Java Byte Code directly

ARM version 5

- ARM DSP extensions:
  - broaden the suitability of the ARM CPU for intensive signal processing (audio, video)
- Jazelle
  - architectural extensions to execute Java Byte Code directly

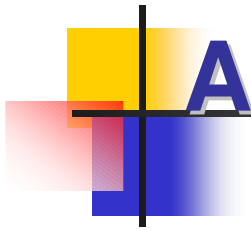


ARM version 5

- ARM DSP extensions:
  - broaden the suitability of the ARM CPU for intensive signal processing (audio, video)
- Jazelle
  - architectural extensions to execute Java Byte Code directly

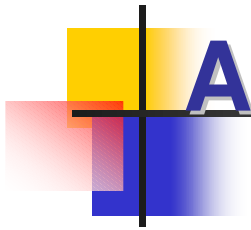
ARM DSP extensions

- Features:
  - single-cycle 16*16 and 32*16 MAC implementations
  - zero overhead saturation extension support
  - new instructions to load/store pairs of registers with enhanced addressing modes
  - new CLZ instruction for normalization in arithmetic operations and improved divide performance



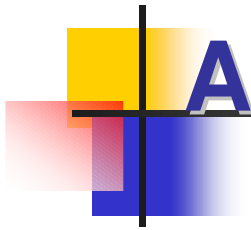
ARM DSP extensions

- Features:
  - single-cycle 16*16 and 32*16 MAC implementations
  - zero overhead saturation extension support
  - new instructions to load/store pairs of registers with enhanced addressing modes
  - new CLZ instruction for normalization in arithmetic operations and improved divide performance



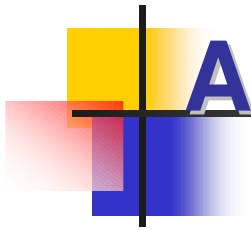
ARM DSP extensions

- Features:
  - single-cycle 16*16 and 32*16 MAC implementations
  - zero overhead saturation extension support
  - new instructions to load/store pairs of registers with enhanced addressing modes
  - new CLZ instruction for normalization in arithmetic operations and improved divide performance



ARM DSP extensions

- Features:
  - single-cycle 16*16 and 32*16 MAC implementations
  - zero overhead saturation extension support
  - new instructions to load/store pairs of registers with enhanced addressing modes
  - new CLZ instruction for normalization in arithmetic operations and improved divide performance

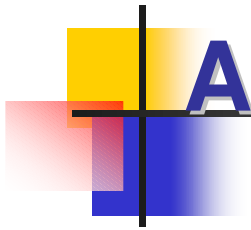


ARM DSP extensions

- Features:
  - single-cycle 16*16 and 32*16 MAC implementations
  - zero overhead saturation extension support
  - new instructions to load/store pairs of registers with enhanced addressing modes
  - new CLZ instruction for normalization in arithmetic operations and improved divide performance

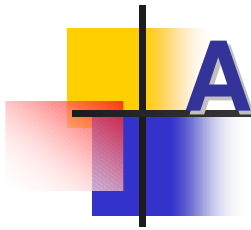
ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM-AMR)



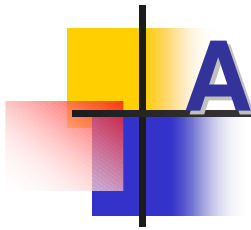
ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM)



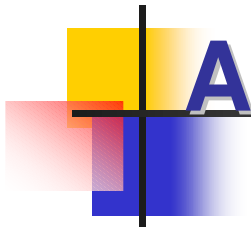
ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM)



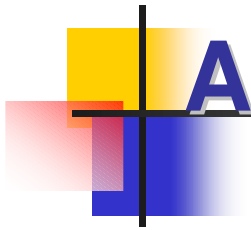
ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM)



ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM)

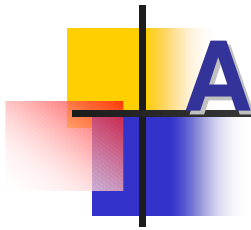


ARM DSP extensions

- Applications:
  - audio encode/decode (AAC,WMA,MP3,..)
  - MPEG4 decode
  - voice and handwriting recognition
  - embedded control
  - bit exact algorithms (GSM)

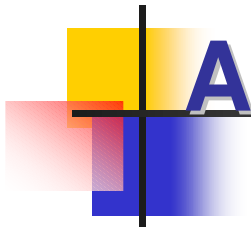
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - hardware logic contribute to 12K gates



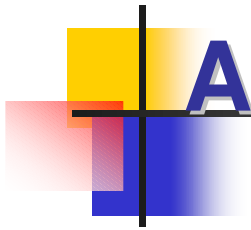
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - hardware logic contribute to 12K gates



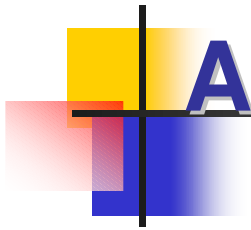
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - hardware logic contribute to 12K gates



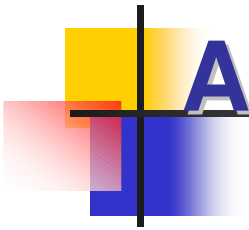
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - hardware logic contribute to 12K gates



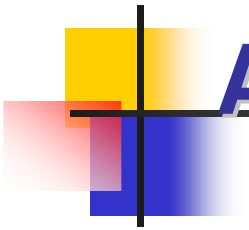
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - hardware logic contribute to 12K gates



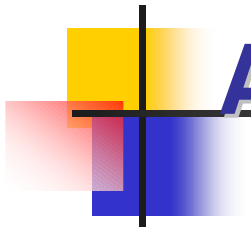
ARM Jazelle

- Specific instruction set
  - to execute directly Java Byte Code
  - reuse of all existing processor resources without the need to re-engineer existing architecture
  - all processor states related to Java execution are stored in normal ARM register set
  - any interrupt routine which saves on entry and restores on exit are compatible with Jazelle
  - additional hardware logic contribute to 12K gates



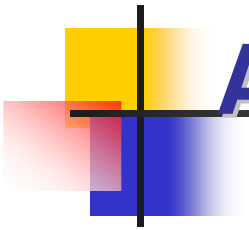
ARM version 6

- media processing extensions
- improved cache architecture
- improved exception and interrupt handling
- unaligned and mixed-endian data support
- six new status bits added to programming model



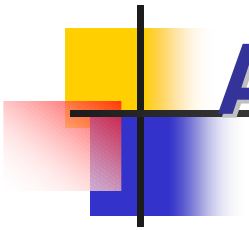
ARM version 6

- media processing extensions
- improved cache architecture
- improved exception and interrupt handling
- unaligned and mixed-endian data support
- six new status bits added to programming model



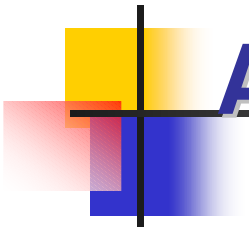
ARM version 6

- media processing extensions
- improved cache architecture
- improved exception and interrupt handling
- unaligned and mixed-endian data support
- six new status bits added to programming model



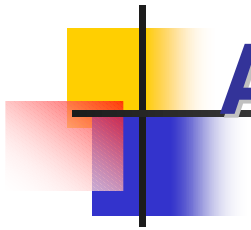
ARM version 6

- media processing extensions
- improved cache architecture
- improved exception and interrupt handling
- unaligned and mixed-endian data support
- six new status bits added to programming model



ARM version 6

- media processing extensions
- improved cache architecture
- improved exception and interrupt handling
- unaligned and mixed-endian data support
- six new status bits added to programming model



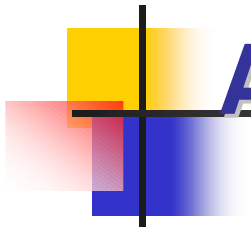
ARMv6 – programming model

- six new status bits

GE[3:0] – SIMD status bits greater than or equal to for each 8/16 bit slice

E-bit : indicates the current load/store endian setting of the core; can be set/cleared with the SETEND instruction

A-bit : indicates if imprecise data abort exceptions are masked



ARMv6 – programming model

- six new status bits

GE[3:0] – SIMD status bits greater than or equal to for each 8/16 bit slice

E-bit : indicates the current load/store endian setting of the core; can be set/cleared with the SETEND instruction

A-bit : indicates if imprecise data abort exceptions are masked

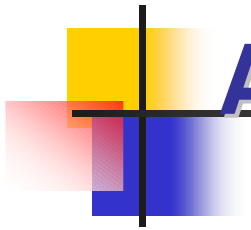
ARMv6 – programming model

- six new status bits

GE[3:0] – SIMD status bits greater than or equal to for each 8/16 bit slice

E-bit : indicates the current load/store endian setting of the core; can be set/cleared with the SETEND instruction

A-bit : indicates if imprecise data abort exceptions are masked



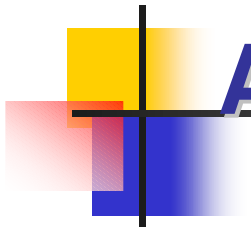
ARMv6 – programming model

- six new status bits

GE[3:0] – SIMD status bits greater than or equal to for each 8/16 bit slice

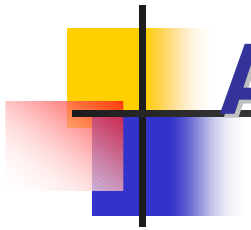
E-bit : indicates the current load/store endian setting of the core; can be set/cleared with the SETEND instruction

A-bit : indicates if imprecise data abort exceptions are masked



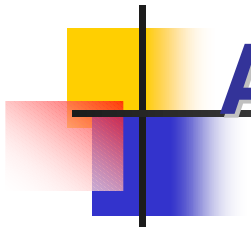
ARMv6 – media instructions

- over 60 SIMD instructions
- enable more efficient software implementation of high-performance media applications
- use the GE-bits added to programming model
- support four 8-bit and two 16-bit operations, parallel add and subtract, selection, packing and unpacking
- support dual 16-bit multiply, add/subtract



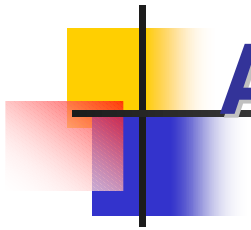
ARMv6 – media instructions

- over 60 SIMD instructions
- enable more efficient software implementation of high-performance media applications
- use the GE-bits added to programming model
- support four 8-bit and two 16-bit operations, parallel add and subtract, selection, packing and unpacking
- support dual 16-bit multiply, add/subtract



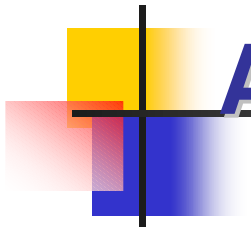
ARMv6 – media instructions

- over 60 SIMD instructions
- enable more efficient software implementation of high-performance media applications
- use the GE-bits added to programming model
- support four 8-bit and two 16-bit operations, parallel add and subtract, selection, packing and unpacking
- support dual 16-bit multiply, add/subtract



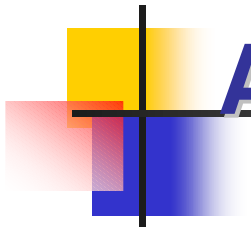
ARMv6 – media instructions

- over 60 SIMD instructions
- enable more efficient software implementation of high-performance media applications
- use the GE-bits added to programming model
- support four 8-bit and two 16-bit operations, parallel add and subtract, selection, packing and unpacking
- support dual 16-bit multiply, add/subtract



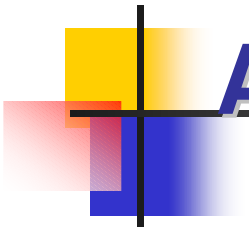
ARMv6 – media instructions

- over 60 SIMD instructions
- enable more efficient software implementation of high-performance media applications
- use the GE-bits added to programming model
- support four 8-bit and two 16-bit operations, parallel add and subtract, selection, packing and unpacking
- support dual 16-bit multiply, add/subtract



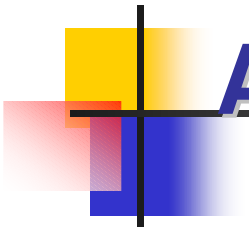
ARMv6 – Thumb 2

- a single Thumb instruction is equivalent to a single ARM instruction
- more Thumb instructions are needed to accomplish the same overall function
- combination of ARM and Thumb code gives better balance of the cost, performance and power characteristics of the system



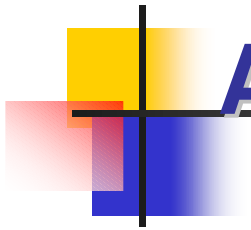
ARMv6 – Thumb 2

- a single Thumb instruction is equivalent to a single ARM instruction
- more Thumb instructions are needed to accomplish the same overall function
- combination of ARM and Thumb code gives better balance of the cost, performance and power characteristics of the system



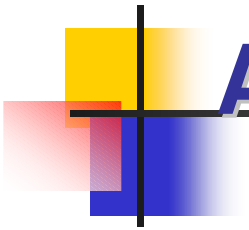
ARMv6 – Thumb 2

- a single Thumb instruction is equivalent to a single ARM instruction
- more Thumb instructions are needed to accomplish the same overall function
- combination of ARM and Thumb code gives better balance of the cost, performance and power characteristics of the system



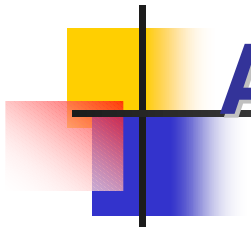
ARMv6 – Thumb 2

- Thumb 2
  - new 16-bit instructions
  - new 32-bit instructions derived from ARM instructions:
    - coprocessor access,
    - privileged instructions
    - special instructions - SIMD



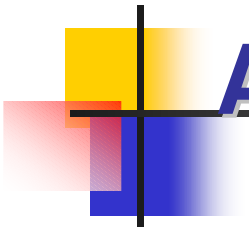
ARMv6 – Thumb 2

- Thumb 2
- new 16-bit instructions
- new 32-bit instructions derived from ARM instructions:
  - coprocessor access,
  - privileged instructions
  - special instructions - SIMD



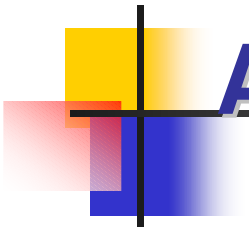
ARMv6 – Thumb 2

- Thumb 2
- new 16-bit instructions
- new 32-bit instructions derived from ARM instructions:
  - coprocessor access,
  - privileged instructions
  - special instructions - SIMD



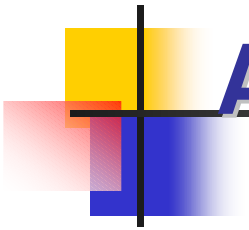
ARMv6 – Thumb 2

- Thumb 2
- new 16-bit instructions
- new 32-bit instructions derived from ARM instructions:
  - coprocessor access
  - privileged instructions
  - special instructions - SIMD



ARMv6 – Thumb 2

- Thumb 2
- new 16-bit instructions
- new 32-bit instructions derived from ARM instructions:
  - coprocessor access
  - privileged instructions
  - special instructions - SIMD



ARMv6 – Thumb 2

- Thumb 2
- new 16-bit instructions
- new 32-bit instructions derived from ARM instructions:
  - coprocessor access,
  - privileged instructions
  - special instructions - SIMD

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

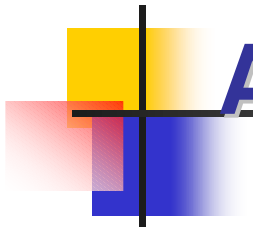
architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

Architecture implementations

architecture	core
V1	ARM1
V2	ARM2
V2A	ARM2AS, ARM3
V3	ARM6, ARM600, ARM610
V3	ARM7, ARM700, ARM710
V4T	ARM7TDMI, ARM710T, ARM720T, ARM740T
V4	StrongARM, ARM8, ARM810
V5	ARM9TDMI, ARM920T, ARM940T
V5TE	ARM9E-S, ARM10TDMI, ARM1020E, XScale
V6	ARM11

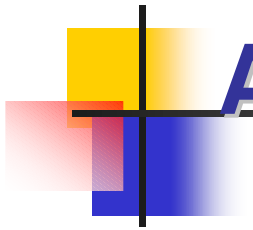


ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points

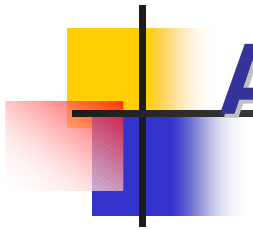


ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points

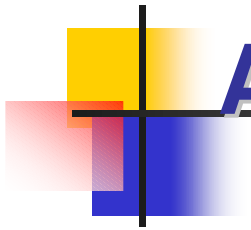


ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points

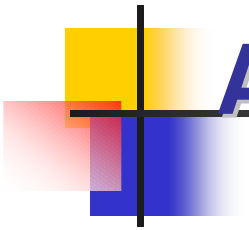


ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points

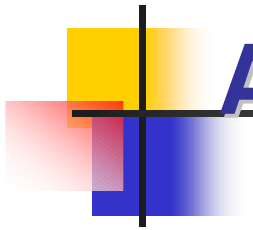


ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points



ARM7TDMI

Evolved from first 32-bit ARM core ARM6

3 volt implementation contains:

- Thumb instruction set
- on-chip Debug support
- an enhanced Multiplier
- embedded ICE hardware for break- and watch- points

ARM7TDMI - pipeline

Using a 3-stage pipeline:

- Implementation of version 4T
- fetch: instruction is fetched from memory and placed in instruction pipeline
- decode: instruction is decoded and datapath control signals prepared
- execute: register bank is read, operand shifted, ALU result generated and written back in a destination register

ARM7TDMI - pipeline

Using a 3-stage pipeline:

- Implementation of version 4T
- fetch: instruction is fetched from memory and placed in instruction pipeline
- decode: instruction is decoded and datapath control signals prepared
- execute: register bank is read, operand shifted, ALU result generated and written back in a destination register

ARM7TDMI - pipeline

Using a 3-stage pipeline:

- Implementation of version 4T
- fetch: instruction is fetched from memory and placed in instruction pipeline
- decode: instruction is decoded and datapath control signals prepared
- execute: register bank is read, operand shifted, ALU result generated and written back in a destination register

ARM7TDMI - pipeline

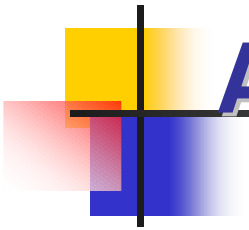
Using a 3-stage pipeline:

- Implementation of version 4T
- fetch: instruction is fetched from memory and placed in instruction pipeline
- decode: instruction is decoded and datapath control signals prepared
- execute: register bank is read, operand shifted, ALU result generated and written back in a destination register

ARM7TDMI - pipeline

Using a 3-stage pipeline:

- Implementation of version 4T
- fetch: instruction is fetched from memory and placed in instruction pipeline
- decode: instruction is decoded and datapath control signals prepared
- execute: register bank is read, operand shifted, ALU result generated and written back in a destination register



ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface

ARM7TDMI - interfaces

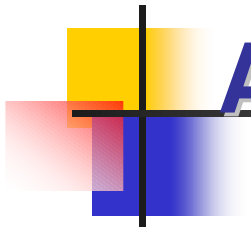
- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface

ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface

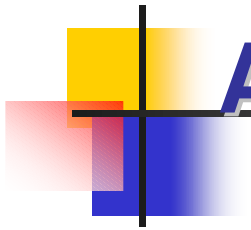
ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface



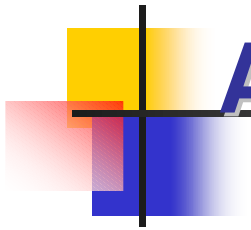
ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface



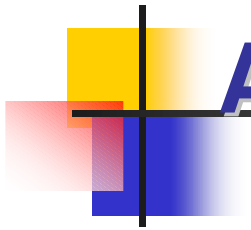
ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface



ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface



ARM7TDMI - interfaces

- register bank: two read ports and one write port
- one additional read and one additional write port for R15
- interfaces:
  - memory interface
  - MMU interface
  - coprocessor interface
  - debug interface
  - JTAG interface

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

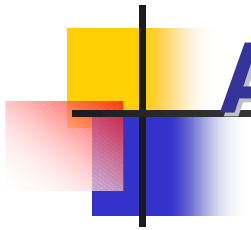
- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690

ARM7TDMI - characteristics

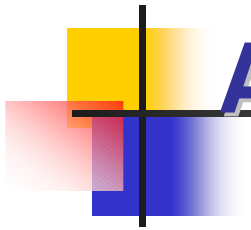
- process: 0,35µm
- metal layers: 3
- Vdd: 3.3 V
- core area: 2.1 mm^2
- power: 87 mW

- transistors: 74 209
- clock: 0-66 MHz
- MIPS: 60
- MIPS/W 690



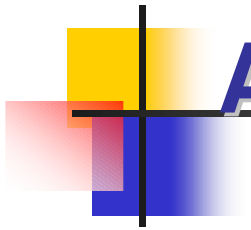
ARM9TDMI - characteristics

- 5-stage pipeline to increase clock rate
- uses separate instruction/data memory ports to improve CPI (Clock cycles Per Instruction)
- Thumb hardware instruction decoding
- static branch prediction
- ARM9E-S is a synthesizable version of the ARM9TDI core



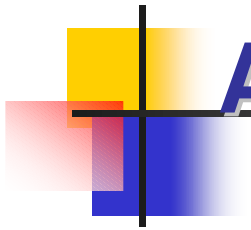
ARM9TDMI - characteristics

- 5-stage pipeline to increase clock rate
- uses separate instruction/data memory ports to improve CPI (Clock cycles Per Instruction)
- Thumb hardware instruction decoding
- static branch prediction
- ARM9E-S is a synthesizable version of the ARM9TDI core



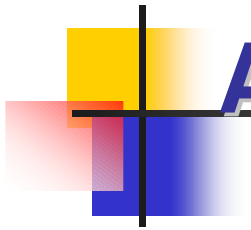
ARM9TDMI - characteristics

- 5-stage pipeline to increase clock rate
- uses separate instruction/data memory ports to improve CPI (Clock cycles Per Instruction)
- Thumb hardware instruction decoding
- static branch prediction
- ARM9E-S is a synthesizable version of the ARM9TDI core



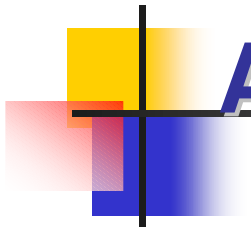
ARM9TDMI - characteristics

- 5-stage pipeline to increase clock rate
- uses separate instruction/data memory ports to improve CPI (Clock cycles Per Instruction)
- Thumb hardware instruction decoding
- static branch prediction
- ARM9E-S is a synthesizable version of the ARM9TDI core



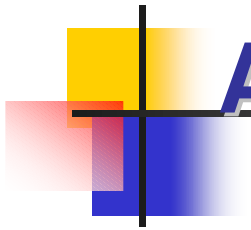
ARM9TDMI - characteristics

- 5-stage pipeline to increase clock rate
- uses separate instruction/data memory ports to improve CPI (Clock cycles Per Instruction)
- Thumb hardware instruction decoding
- static branch prediction
- ARM9E-S is a synthesizable version of the ARM9TDI core



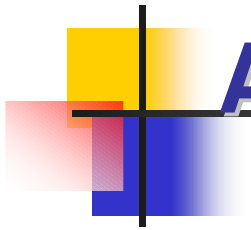
ARM9TDMI - StrongARM

- StrongARM has a dedicated branch adder which operates in parallel with the register read stage
- ARM9TDMI uses the main ALU – an additional clock cycle penalty for a taken branch but smaller core
- StrongARM designed for particular technology
- ARM9TDMI is readily portable to a new process



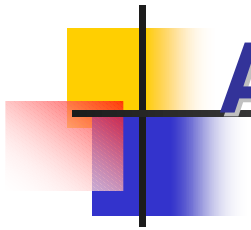
ARM9TDMI - StrongARM

- StrongARM has a dedicated branch adder which operates in parallel with the register read stage
- ARM9TDMI uses the main ALU – an additional clock cycle penalty for a taken branch but smaller core
- StrongARM designed for particular technology
- ARM9TDMI is readily portable to a new process



ARM9TDMI - StrongARM

- StrongARM has a dedicated branch adder which operates in parallel with the register read stage
- ARM9TDMI uses the main ALU – an additional clock cycle penalty for a taken branch but smaller core
- StrongARM designed for particular technology
- ARM9TDMI is readily portable to a new process



ARM9TDMI - StrongARM

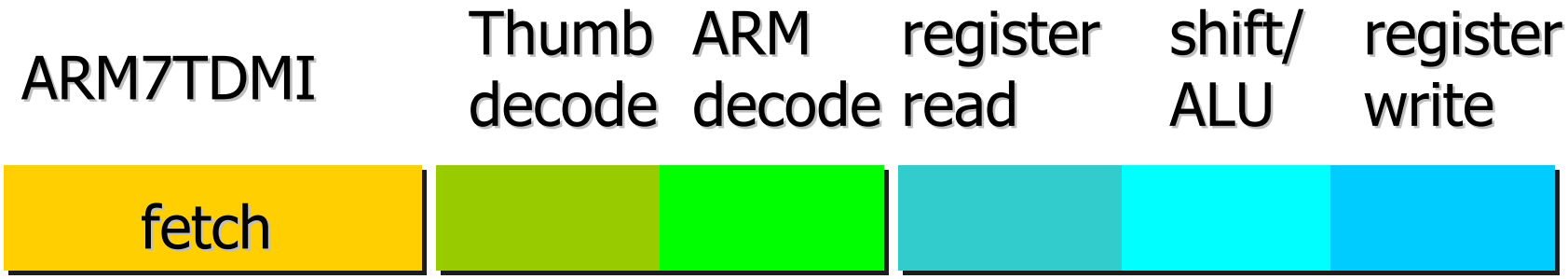
- StrongARM has a dedicated branch adder which operates in parallel with the register read stage
- ARM9TDMI uses the main ALU – an additional clock cycle penalty for a taken branch but smaller core
- StrongARM designed for particular technology
- ARM9TDMI is readily portable to a new process

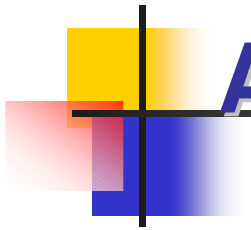
ARM7TDMI pipeline

ARM7TDMI

ARM7TDMI pipeline

ARM7TDMI pipeline



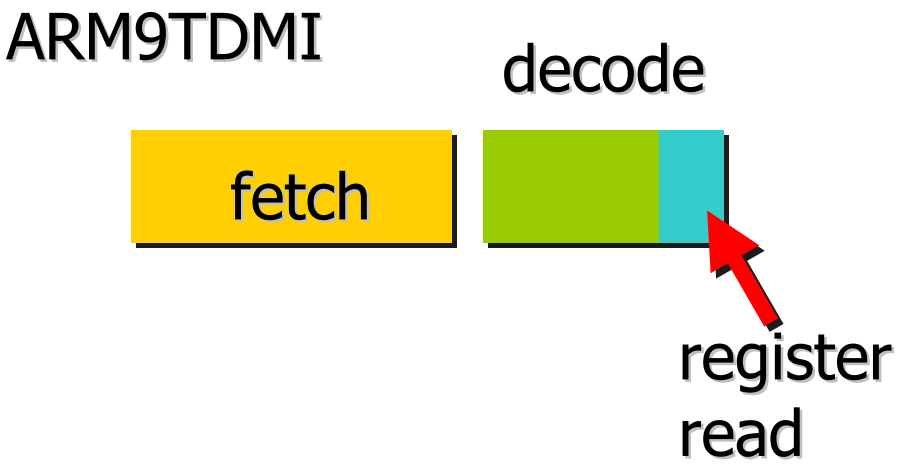


ARM9TDMI pipeline

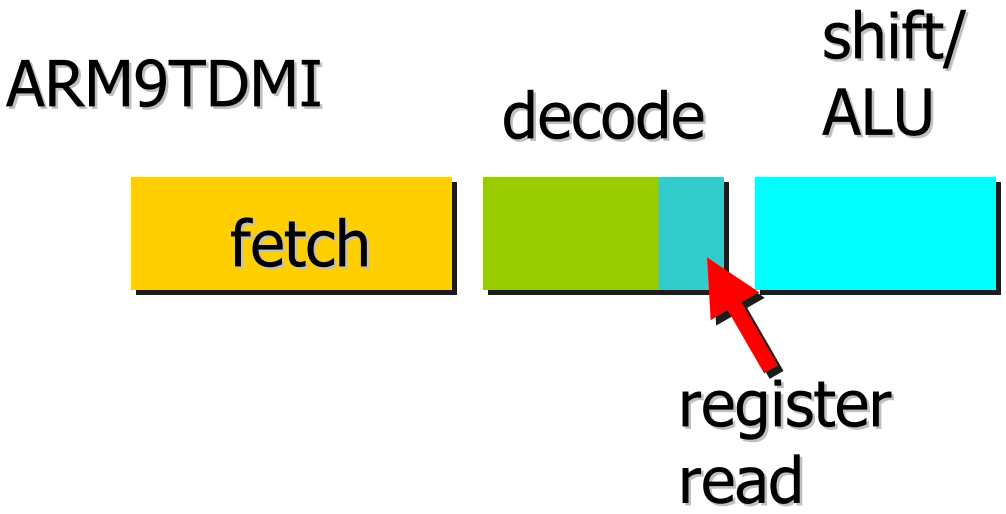
ARM9TDMI

fetch

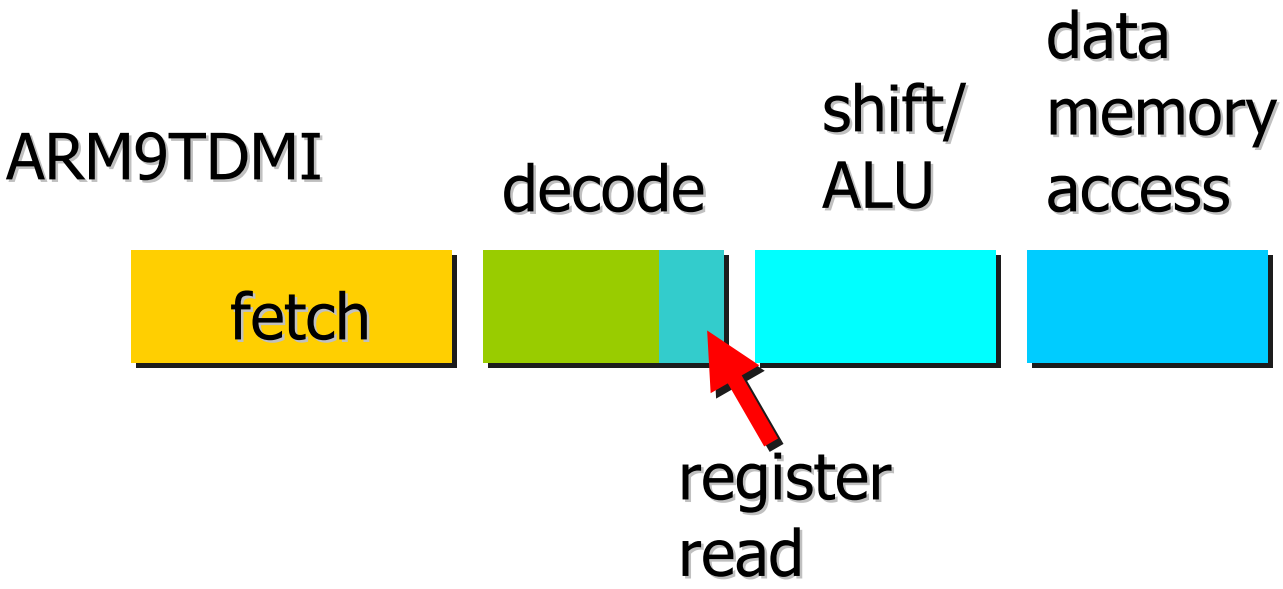
ARM9TDMI pipeline



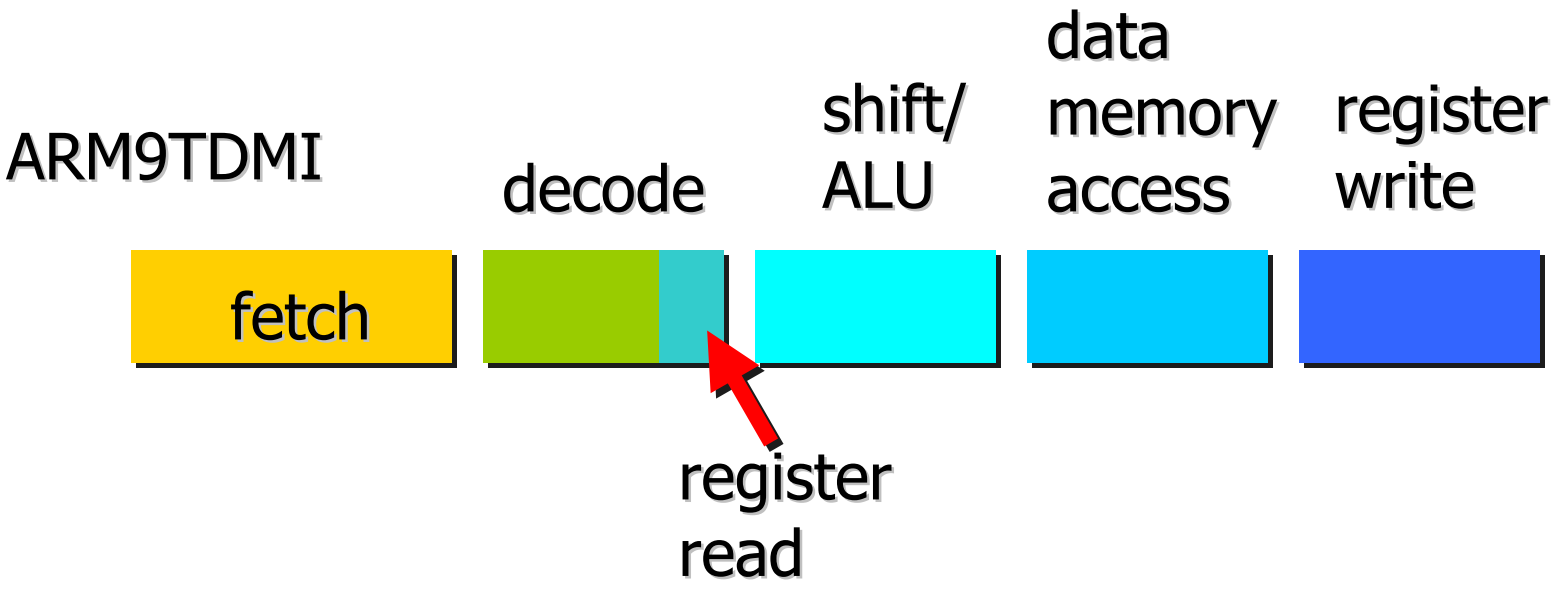
ARM9TDMI pipeline

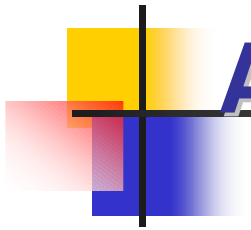


ARM9TDMI pipeline



ARM9TDMI pipeline

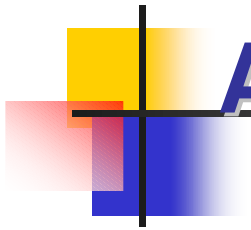




ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

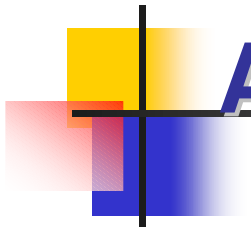
- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



ARM9TDMI characteristics

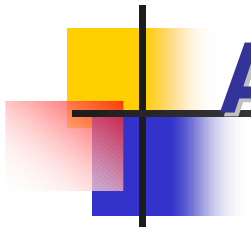
- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500

ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

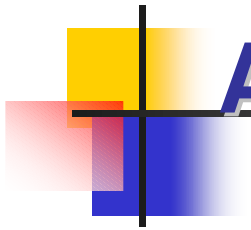
- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

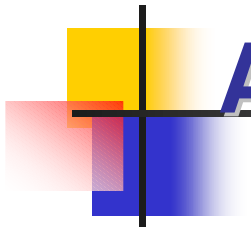
- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



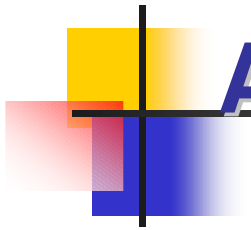
ARM9TDMI characteristics

- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500

ARM9TDMI characteristics

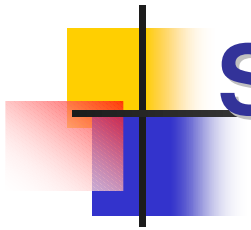
- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW
- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



ARM9TDMI characteristics

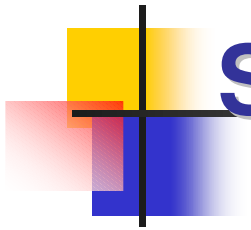
- process: 0,25µm
- metal layers: 3
- Vdd: 2.5 V
- core area: 2.1 mm^2
- power: 150 mW

- transistors: 111 000
- clock: 0-200 MHz
- MIPS: 220
- MIPS/W 1500



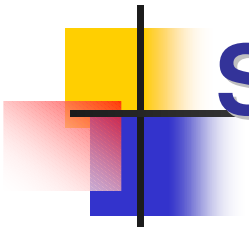
Summary

- ARM pre-history
- licensing ARM
- ARM architecture definition
- ARM versions
- ARM architecture implementations



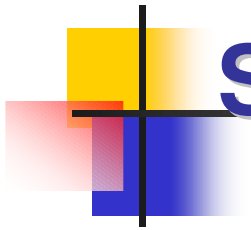
Summary

- ARM pre-history
- licensing ARM
- ARM architecture definition
- ARM versions
- ARM architecture implementations



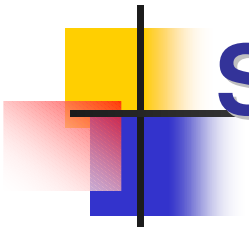
Summary

- ARM pre-history
- licensing ARM
- ARM architecture definition
- ARM versions
- ARM architecture implementations



Summary

- ARM pre-history
- licensing ARM
- ARM architecture definition
- ARM versions
- ARM architecture implementations



Summary

- ARM pre-history
- licensing ARM
- ARM architecture definition
- ARM versions
- ARM architecture implementations