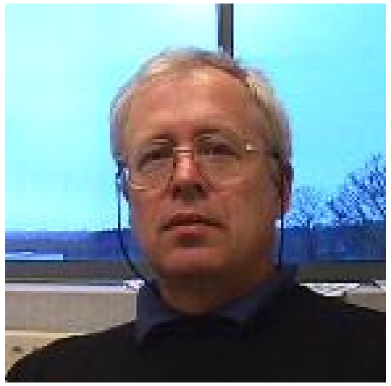
An introduction to ARM processor architecture

P. Bakowski



bako@ieee.org

From ACORN to ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

From ACORN to ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

Berkeley RISC I and ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

Berkeley RISC I and ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

Berkeley RISC I and ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

Berkeley RISC I and ARM

- Acorn Computers Limited – Cambridge 1985
- Advanced Risc Machine – Cambridge 1990

Berkeley RISC I and ARM processor

- a load-store architecture
- fixed-length 32-bit instructions
- 3-address instruction formats

Berkeley RISC I and ARM

Rejected features of Berkeley RISC

- register windows
- delayed branches
- single execution stage for all instructions

Berkeley RISC I and ARM

Rejected features of Berkeley RISC

- register windows
- delayed branches
- single execution stage for all instructions

Berkeley RISC I and ARM

Rejected features of Berkeley RISC

- register windows
- delayed branches
- single execution stage for all instructions

Berkeley RISC I and ARM

Rejected features of Berkeley RISC

- register windows
- delayed branches
- single execution stage for all instructions

ARM architecture versions

The ARM instruction set architecture has evolved significantly since it was first developed, and will continue to be developed in the future.

In order to be precise about which instructions exist in any particular ARM implementation, five major versions of the instruction set have been defined to date.

These are denoted by the version numbers 1 to 6.

ARM architecture versions

The ARM instruction set architecture has evolved significantly since it was first developed, and will continue to be developed in the future.

In order to be precise about which instructions exist in any particular ARM implementation, five major versions of the instruction set have been defined to date.

These are denoted by the version numbers 1 to 6.

ARM architecture versions

The ARM instruction set architecture has evolved significantly since it was first developed, and will continue to be developed in the future.

In order to be precise about which instructions exist in any particular ARM implementation, five major versions of the instruction set have been defined to date.

These are denoted by the version numbers 1 to 6.

ARM architecture versions

Many of the versions can be qualified with variant letters to specify collections of additional instructions that are included in that version.

ARM architecture versions

Many of the versions can be qualified with variant letters to specify collections of additional instructions that are included in that version.

These collections vary from being

very small (the M variant denotes the addition of just four extra instructions) to

very large (the T variant denotes the addition of the entire Thumb instruction set).

ARM architecture versions

Many of the versions can be qualified with variant letters to specify collections of additional instructions that are included in that version.

These collections vary from being

• very small (the M variant denotes the addition of just four extra instructions) to

• very large (the T variant denotes the addition of the entire Thumb instruction set)

ARM architecture versions

Many of the versions can be qualified with variant letters to specify collections of additional instructions that are included in that version.

These collections vary from being

• very small (the M variant denotes the addition of just four extra instructions) to

• very large (the T variant denotes the addition of the entire Thumb instruction set)

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

• the basic data-processing instructions (not including multiplies)

• byte, word, and multi-word load/store instructions

• branch instructions, including a branch-and-link instruction designed for subroutine calls

• a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

• the basic data-processing instructions (not including multiplies)

• byte, word, and multi-word load/store instructions

• branch instructions, including a branch-and-link instruction designed for subroutine calls

• a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

- the basic data-processing instructions (not including multiplies)
- byte, word, and multi-word load/store instructions
- branch instructions, including a branch-and-link instruction designed for subroutine calls
- a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

• the basic data-processing instructions (not including multiplies)

• byte, word, and multi-word load/store instructions

• branch instructions, including a branch-and-link instruction designed for subroutine calls

• a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

• the basic data-processing instructions (not including multiplies)

• byte, word, and multi-word load/store instructions

• branch instructions, including a branch-and-link instruction designed for subroutine calls

• a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 1

Version 1 was implemented only by ARM1, and was never used in a commercial product. It contained:

- the basic data-processing instructions (not including multiplies)
- byte, word, and multi-word load/store instructions
- branch instructions, including a branch-and-link instruction designed for subroutine calls
- a software interrupt instruction, for use in making Operating System calls.

Version 1 only had a 26-bit address space, and is now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

• multiply and multiply-accumulate instructions

• coprocessor support

• two more banked registers in fast interrupt mode

• atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

- multiply and multiply-accumulate instructions
- coprocessor support
- two more banked registers in fast interrupt mode
- atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

- multiply and multiply-accumulate instructions
- coprocessor support
- two more banked registers in fast interrupt mode
- atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

- multiply and multiply-accumulate instructions
- coprocessor support
- two more banked registers in fast interrupt mode
- atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

- multiply and multiply-accumulate instructions
- coprocessor support
- two more banked registers in fast interrupt mode
- atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 2

Version 2 is extended architecture version 1 by adding:

- multiply and multiply-accumulate instructions
- coprocessor support
- two more banked registers in fast interrupt mode
- atomic load-and-store instructions called SWP and SWPB (in a slightly later variant called version 2a)

Version 2 and 2a still only had a 26-bit address space, and are now obsolete.

ARM architecture version 3

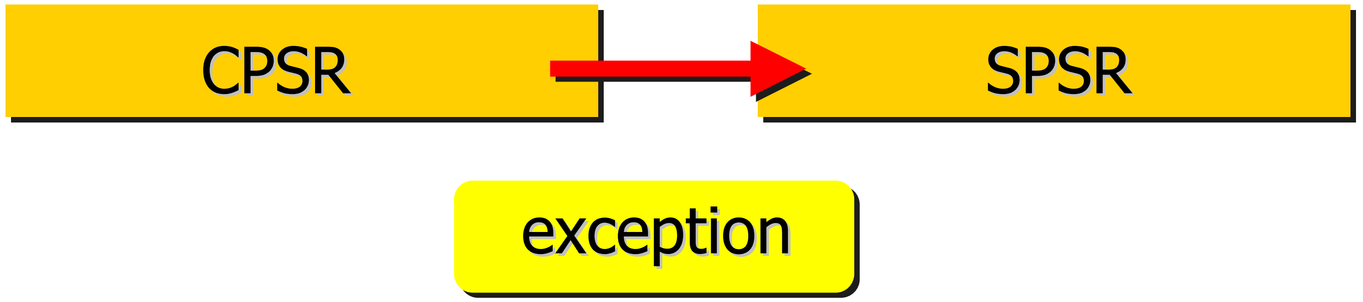
Version 3 extended the addressing range to 32 bits.

Program status information which had previously been stored in R15 was moved to a new Current Program Status Register (CPSR), and Saved Program Status Registers (SPSRs) where added to preserve the CPSR contents when an exceptions occurred.

ARM architecture version 3

Version 3 extended the addressing range to 32 bits.

Program status information which had previously been stored in R15 was moved to a new Current Program Status Register (CPSR), and Saved Program Status Registers (SPSRs) where added to preserve the CPSR contents when an exceptions occurred.



ARM architecture version 3

As a result, the following changes occurred to the instruction set:

- two instructions (MRS and MSR) were added to allow the new CPSR and SPSRs to be accessed.
- the functionality of instructions previously used to return from exceptions was modified to allow them to continue to be used for that purpose.

ARM architecture version 3

As a result, the following changes occurred to the instruction set:

- two instructions (MRS and MSR) were added to allow the new CPSR and SPSRs to be accessed.
- the functionality of instructions previously used to return from exceptions was modified to allow them to continue to be used for that purpose.

ARM architecture version 3

As a result, the following changes occurred to the instruction set:

- two instructions (MRS and MSR) were added to allow the new CPSR and SPSRs to be accessed.
- the functionality of instructions previously used to return from exceptions was modified to allow them to continue to be used for that purpose.

ARM architecture version 3

Version 3 also added two new processor modes in order to make it possible to use Data Abort, Prefetch Abort and Undefined Instruction exceptions effectively in Operating System code.

Backwards-compatibility support for the 26-bit architectures was obligatory in version 3, except in a variant called version 3G.

The distinction between version 3 and 3G is now obsolete.

ARM architecture version 3

Version 3 also added two new processor modes in order to make it possible to use Data Abort, Prefetch Abort and Undefined Instruction exceptions effectively in Operating System code.

Backwards-compatibility support for the 26-bit architectures was obligatory in version 3, except in a variant called version 3G.

The distinction between version 3 and 3G is now obsolete.

ARM architecture version 3

Version 3 also added two new processor modes in order to make it possible to use Data Abort, Prefetch Abort and Undefined Instruction exceptions effectively in Operating System code.

Backwards-compatibility support for the 26-bit architectures was obligatory in version 3, except in a variant called version 3G.

The distinction between version 3 and 3G is now obsolete.

Overview of 26-bit architecture

ARM v1, ARM v2, and ARM v2a are earlier versions of the ARM architecture which implemented only a 26-bit address space, and are known as 26-bit architectures.

ARM architecture version 3 and above implement a 32-bit address space and are known as 32-bit architectures.

For backwards compatibility, except for ARMv3G, all variants of ARM architecture version 3 implement the 26-bit address space.

All not-T variants of ARM architecture version 4 and above can optionally implement the 26-bit address space.

Overview of 26-bit architecture

ARM v1, ARM v2, and ARM v2a are earlier versions of the ARM architecture which implemented only a 26-bit address space, and are known as 26-bit architectures.

ARM architecture version 3 and above implement a 32-bit address space and are known as 32-bit architectures.

For backwards compatibility, except for ARMv3G, all variants of ARM architecture version 3 implement the 26-bit address space.

All not-T variants of ARM architecture version 4 and above can optionally implement the 26-bit address space.

Overview of 26-bit architecture

ARM v1, ARM v2, and ARM v2a are earlier versions of the ARM architecture which implemented only a 26-bit address space, and are known as 26-bit architectures.

ARM architecture version 3 and above implement a 32-bit address space and are known as 32-bit architectures.

For backwards compatibility, except for ARMv3G, all variants of ARM architecture version 3 implement the 26-bit address space.

All not-T variants of ARM architecture version 4 and above can optionally implement the 26-bit address space.

Overview of 26-bit architecture

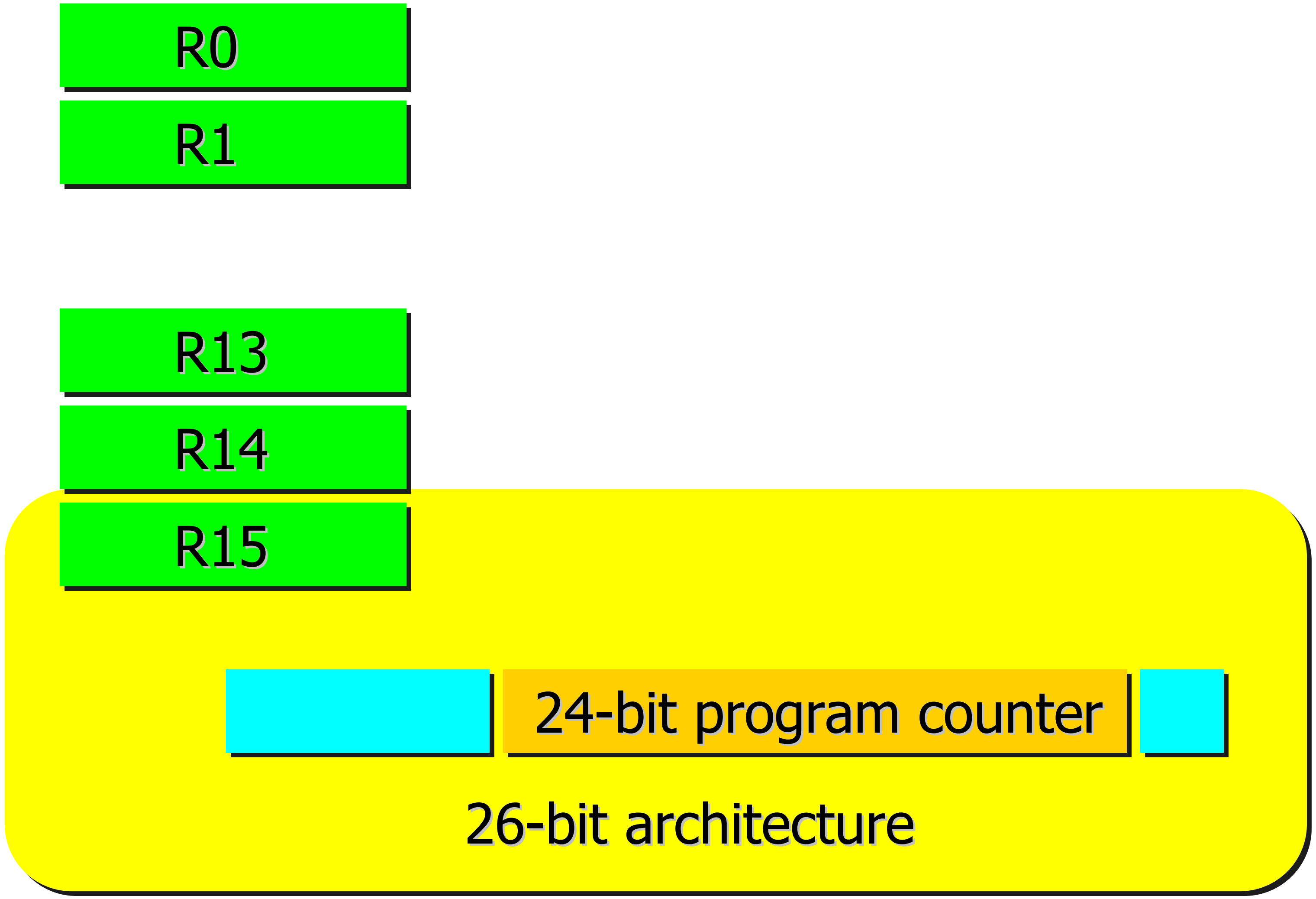
ARM v1, ARM v2, and ARM v2a are earlier versions of the ARM architecture which implemented only a 26-bit address space, and are known as 26-bit architectures.

ARM architecture version 3 and above implement a 32-bit address space and are known as 32-bit architectures.

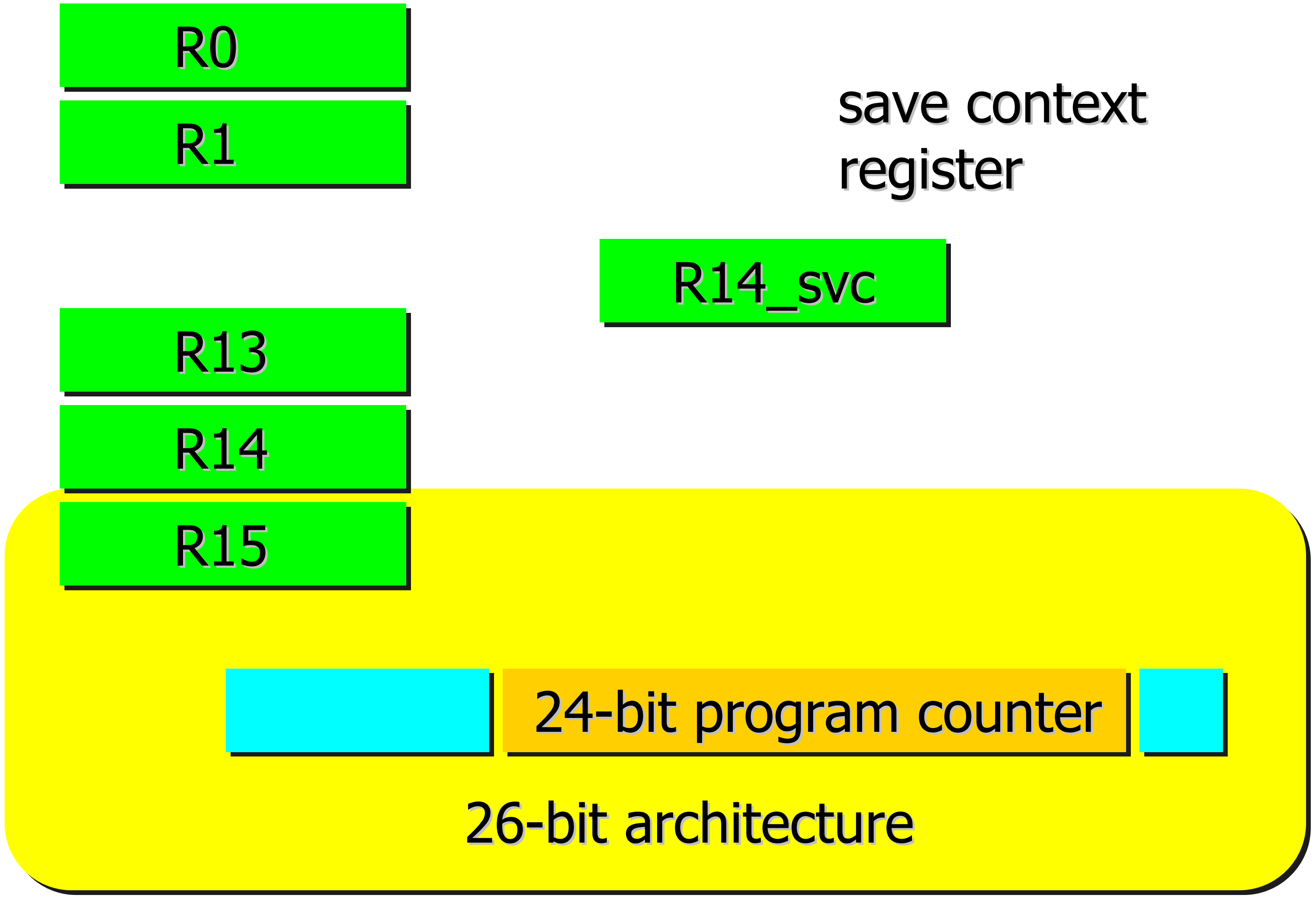
For backwards compatibility, except for ARMv3G, all variants of ARM architecture version 3 implement the 26-bit address space.

All not-T variants of ARM architecture version 4 and above can optionally implement the 26-bit address space.

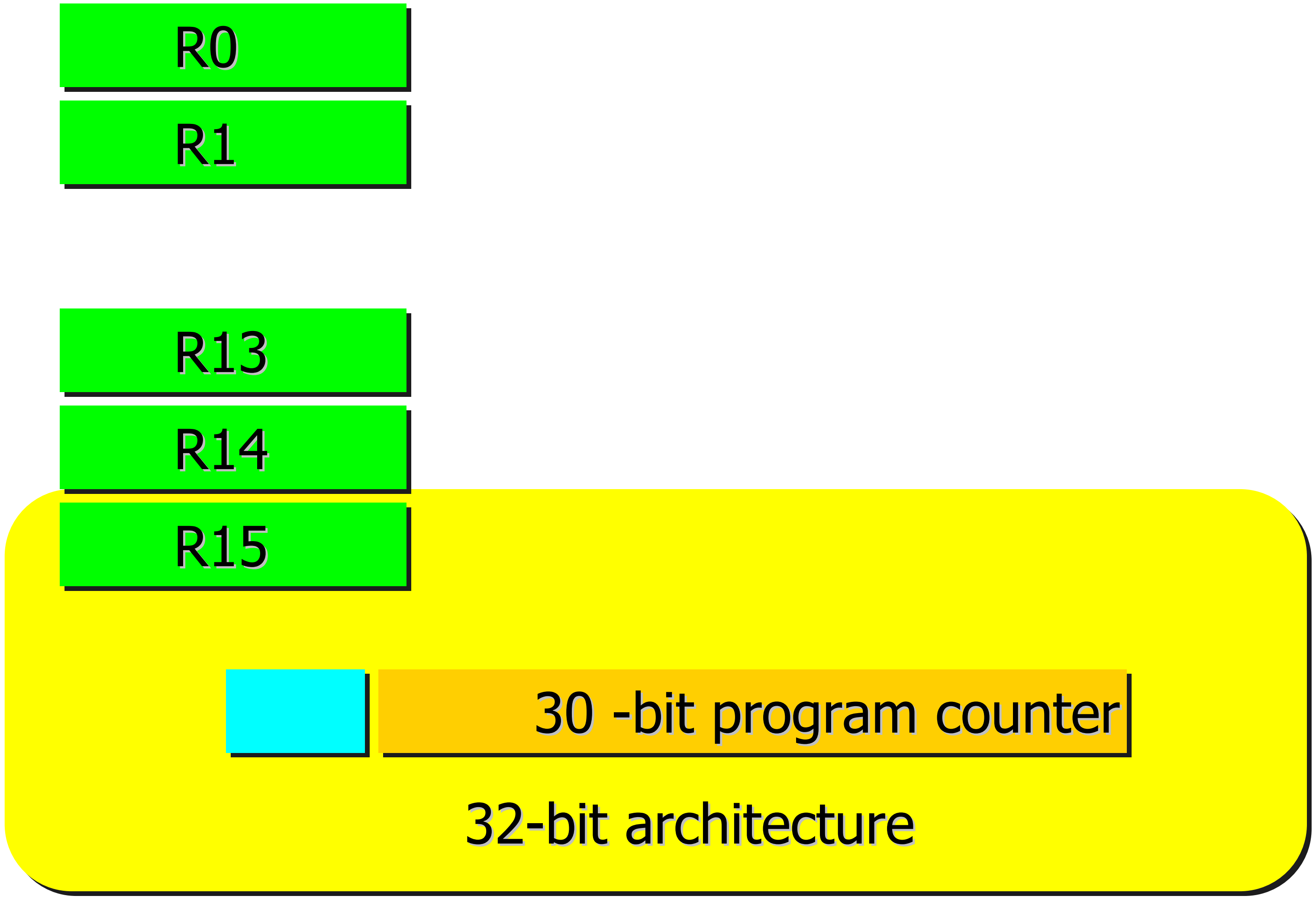
ARM registers: 26-bit



ARM registers: 26-bit



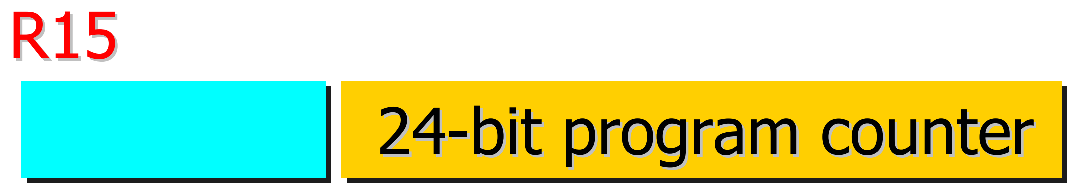
ARM registers: 32-bit



26-bit : program counter

The 26-bit architecture implement only a 24-bit program counter in R15, which allows 64MB of program space.

The 32-bit architecture have a 30-bit program counter in R15 witch allows 4GB of program space on 32-bit architectures.

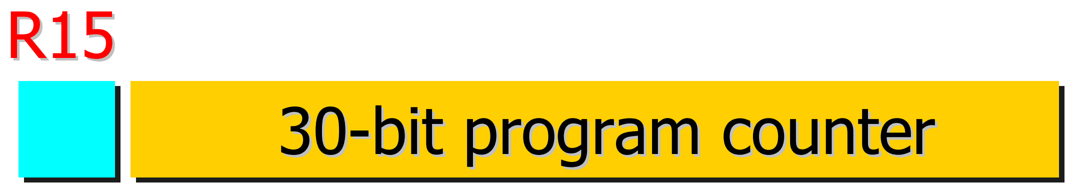


26-bit architecture

32-bit : program counter

The 26-bit architecture implement only a 24-bit program counter in R15, which allows 64MB of program space.

The 32-bit architecture have a 30-bit program counter in R15 witch allows 4GB of program space on 32-bit architectures.

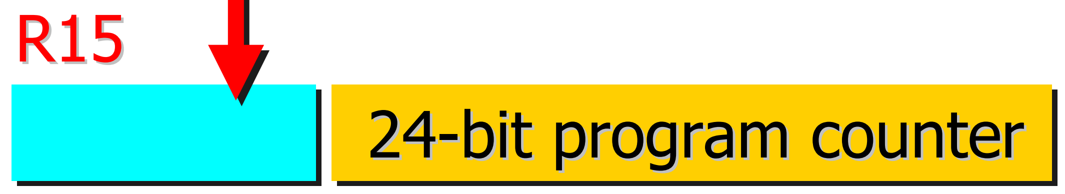


32-bit architecture

26-bit : processor modes

Only four processor modes are supported on 26-bit architectures:

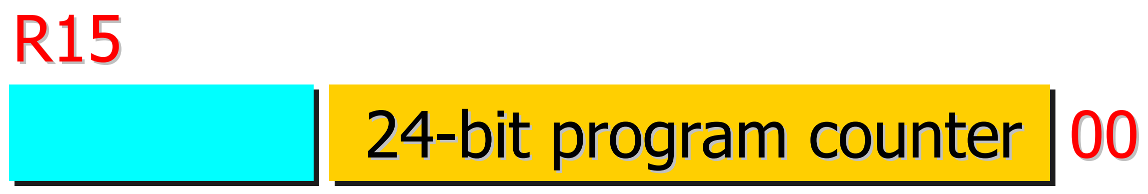
- user (0b00)
- FIQ (0b01)
- IRQ (0b10)
- supervisor (0b11)



26-bit : processor modes

Only four processor modes are supported on 26-bit architectures:

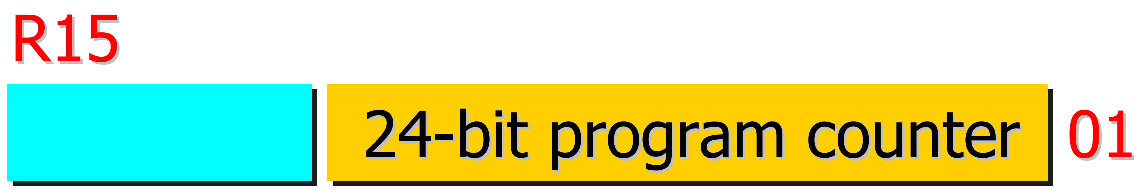
- user (0b00)
- FIQ (0b01)
- IRQ (0b10)
- supervisor (0b11)



26-bit : processor modes

Only four processor modes are supported on 26-bit architectures:

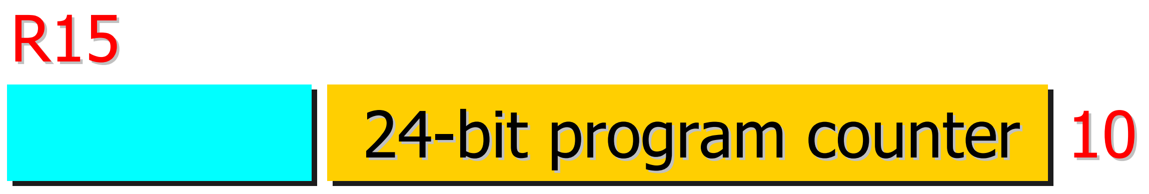
- user (0b00)
- FIQ (0b01) – fast interruption request
- IRQ (0b10)
- supervisor (0b11)



26-bit : processor modes

Only four processor modes are supported on 26-bit architectures:

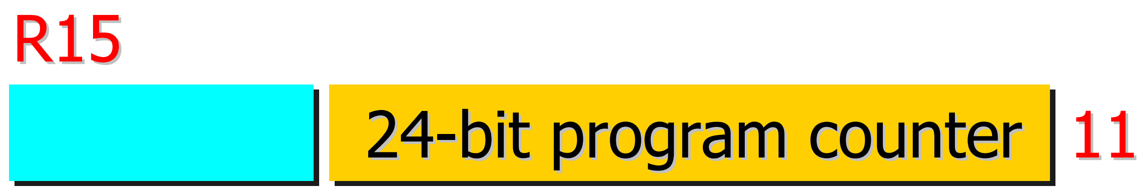
- user (0b00)
- FIQ (0b01)
- IRQ (0b10) – interruption request
- supervisor (0b11)



26-bit : processor modes

Only four processor modes are supported on 26-bit architectures:

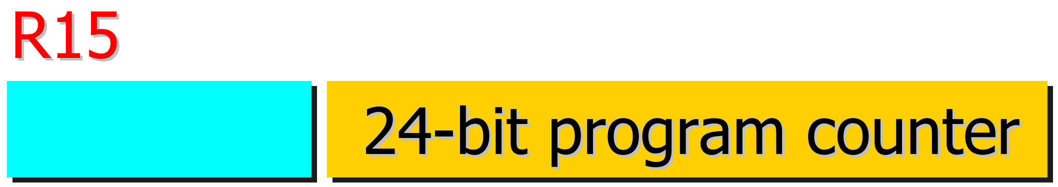
- user (0b00)
- FIQ (0b01)
- IRQ (0b10)
- supervisor (0b11)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)

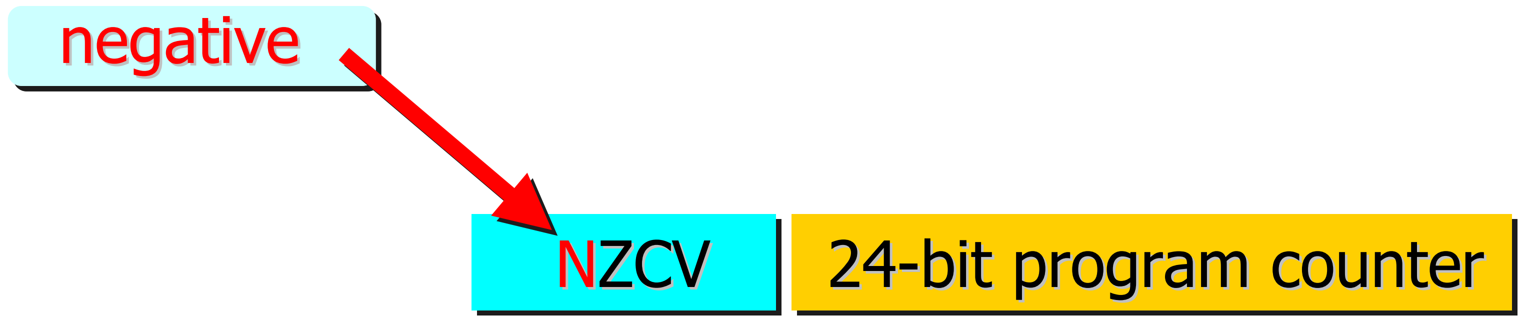


P. Bakowski

26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

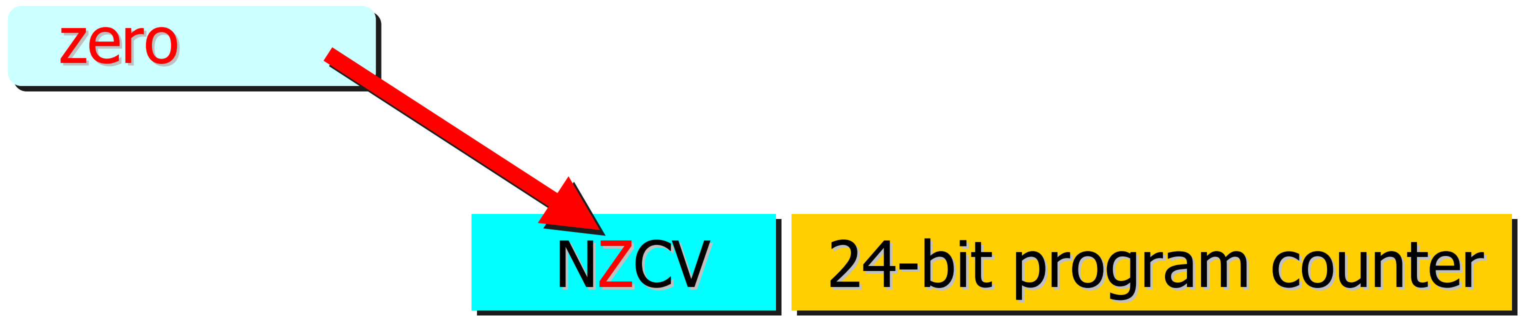
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

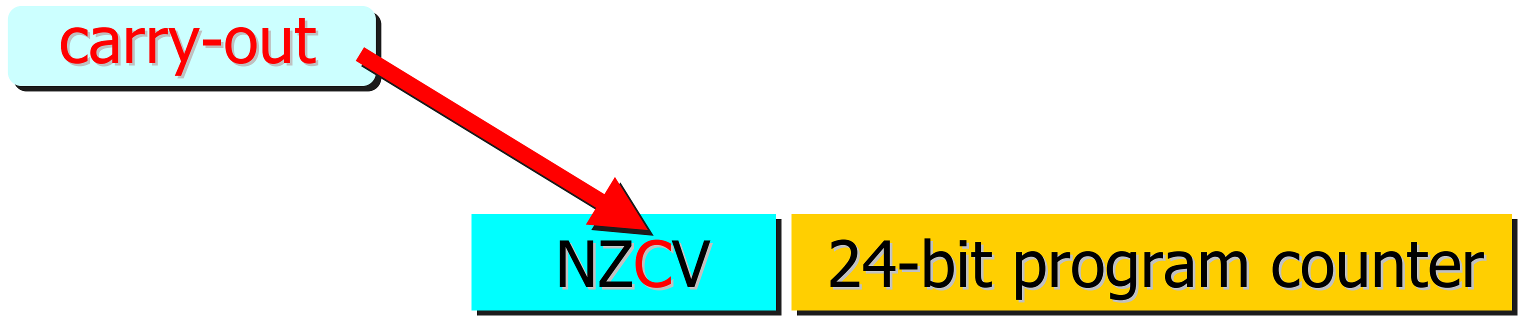
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

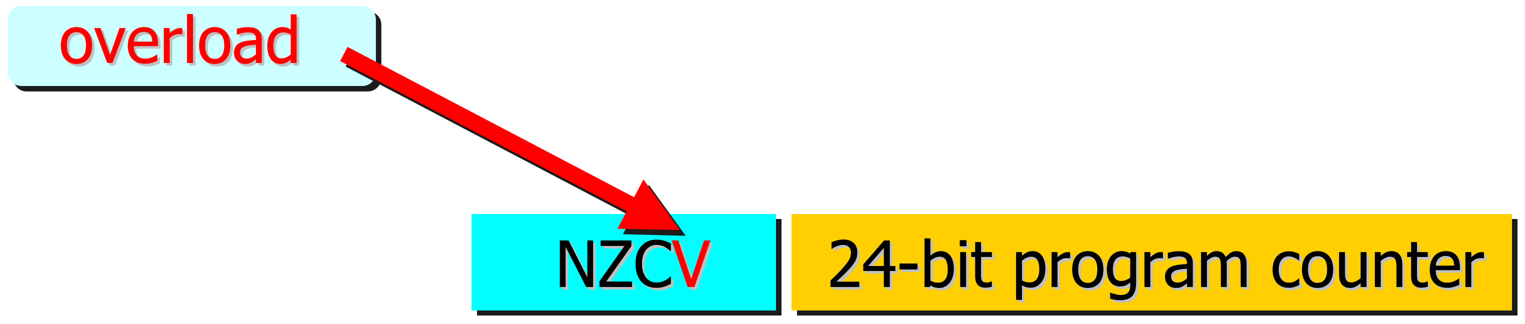
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

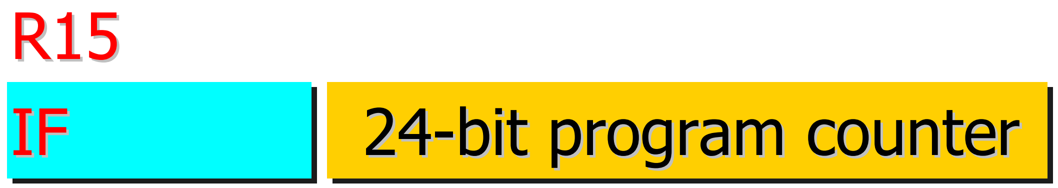
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

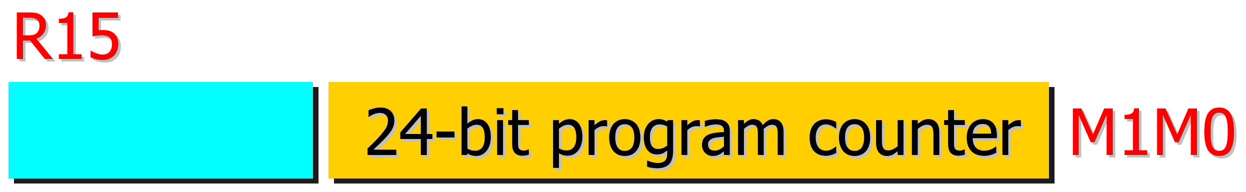
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

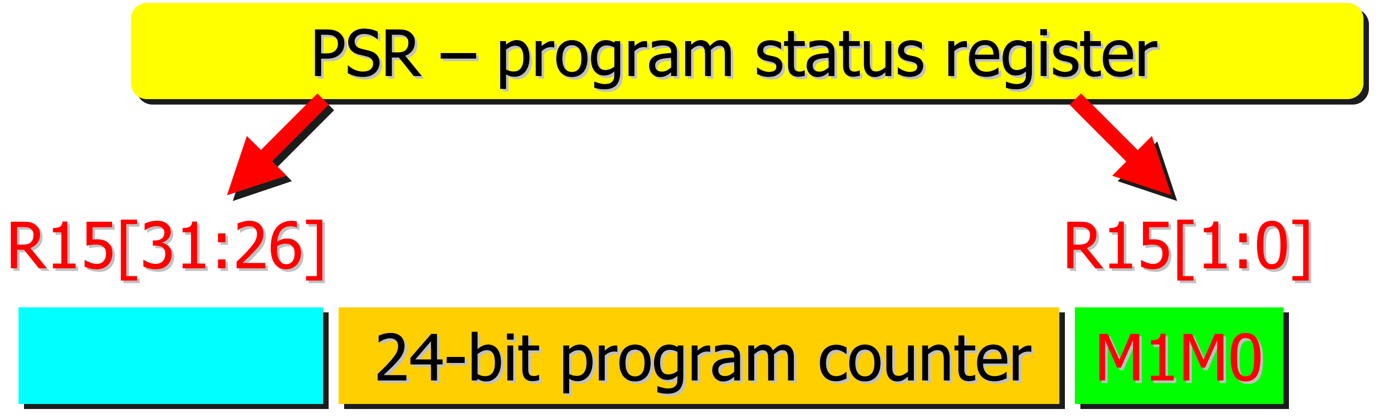
- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor modes & flags

In the 26-bit architectures, the following are also stored in register 15 :

- four condition flags (N,Z,C and V)
- the interrupt disable flags (I and F)
- two processor modes bits (M1 and M0)



26-bit : processor exception

The precise effect of an exception on a 26-bit architecture is the following:

• the banked version of R14 has bits [25:2] set to the specified address, and bits [31:26, 1, 0] set to the copies of the corresponding bits in R15.

• the I, F, M1, and M0 bits are modified in the same way as CPSR[7], CPSR[6], CPSR[1], and CPSR[0] respectively, on a 32-bit architecture.

The I, F, M1, and M0 bits cannot be written directly when the processor is in User mode. In User mode they are only changed by an exception occurring.

26-bit : processor exception

The precise effect of an exception on a 26-bit architecture is the following:

- the banked version of R14 has bits [25:2] set to the specified address, and bits [31:26, 1, 0] set to the copies of the corresponding bits in R15.
- the I, F, M1, and M0 bits are modified in the same way as CPSR[7], CPSR[6], CPSR[1], and CPSR[0] respectively, on a 32-bit architecture.

The I, F, M1, and M0 bits cannot be written directly when the processor is in User mode. In User mode they are only changed by an exception occurring.

26-bit : processor exception

The precise effect of an exception on a 26-bit architecture is the following:

- the banked version of R14 has bits [25:2] set to the specified address, and bits [31:26, 1, 0] set to the copies of the corresponding bits in R15.
- the I, F, M1, and M0 bits are modified in the same way as CPSR[7], CPSR[6], CPSR[1], and CPSR[0] respectively, on a 32-bit architecture.

The I, F, M1, and M0 bits cannot be written directly when the processor is in User mode. In User mode they are only changed by an exception occurring.

26-bit : processor exception

The precise effect of an exception on a 26-bit architecture is the following:

• the banked version of R14 has bits [25:2] set to the specified address, and bits [31:26, 1, 0] set to the copies of the corresponding bits in R15.

• the I, F, M1, and M0 bits are modified in the same way as CPSR[7], CPSR[6], CPSR[1], and CPSR[0] respectively, on a 32-bit architecture.

The I, F, M1, and M0 bits cannot be written directly when the processor is in User mode. In User mode they are only changed by an exception occurring.

26-bit : reading register 15

In 26-bit architecture the value of R15 is read in four different ways:

- if R15 is specified in bits [19:16] of an instruction, only the PC (bits [25:2]) is used. All other bits read as zero.
- if R15 is specified in bits [3:0] of an instruction, all 32 bits are used.
- if R15 is stored using STR or STM, the value of the PC (bits [25:2]) is implementation defined but all 32 bits of the register are stored.
- all 32 bits are stored in the Link register (R14) after a Branch with Link instruction or an exception entry.

26-bit : reading register 15

In 26-bit architecture the value of R15 is read in four different ways:

- if R15 is specified in bits [19:16] of an instruction, only the PC (bits [25:2]) is used. All other bits read as zero.
- if R15 is specified in bits [3:0] of an instruction, all 32 bits are used.
- if R15 is stored using STR or STM, the value of the PC (bits [25:2]) is implementation defined but all 32 bits of the register are stored.
- all 32 bits are stored in the Link register (R14) after a Branch with Link instruction or an exception entry.

26-bit : reading register 15

In 26-bit architecture the value of R15 is read in four different ways:

• if R15 is specified in bits [19:16] of an instruction, only the PC (bits [25:2]) is used. All other bits read as zero.

• if R15 is specified in bits [3:0] of an instruction, all 32 bits are used.

• if R15 is stored using STR or STM, the value of the PC (bits [25:2]) is implementation defined but all 32 bits of the register are stored.

• all 32 bits are stored in the Link register (R14) after a Branch with Link instruction or an exception entry.

26-bit : reading register 15

In 26-bit architecture the value of R15 is read in four different ways:

• if R15 is specified in bits [19:16] of an instruction, only the PC (bits [25:2]) is used. All other bits read as zero.

• if R15 is specified in bits [3:0] of an instruction, all 32 bits are used.

• if R15 is stored using STR or STM, the value of the PC (bits [25:2]) is implementation defined but all 32 bits of the register are stored.

• all 32 bits are stored in the Link register (R14) after a Branch with Link instruction or an exception entry.

26-bit : reading register 15

In 26-bit architecture the value of R15 is read in four different ways:

- if R15 is specified in bits [19:16] of an instruction, only the PC (bits [25:2]) is used. All other bits read as zero.
- if R15 is specified in bits [3:0] of an instruction, all 32 bits are used.
- if R15 is stored using STR or STM, the value of the PC (bits [25:2]) is implementation defined but all 32 bits of the register are stored.
- all 32 bits are stored in the Link register (R14) after a Branch with Link instruction or an exception entry.

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions only write the PC part of R15, leaving the PSR part unchanged:

- Data-processing instructions without the S bit set

- LDR instructions

- LDM instructions, other than Load Multiple with Restore CPSR

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions only write the PC part of R15, leaving the PSR part unchanged:

- Data-processing instructions without the S bit set

- LDR instructions

- LDM instructions, other than Load Multiple with Restore CPSR

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions only write the PC part of R15, leaving the PSR part unchanged:

- Data-processing instructions without the S bit set

- LDR instructions

- LDM instructions, other than Load Multiple with Restore CPSR

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions only write the PC part of R15, leaving the PSR part unchanged:

- Data-processing instructions without the S bit set

- LDR instructions

- LDM instructions, other than Load Multiple with Restore CPSR

32-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions only write the PC part of R15, leaving the PSR part unchanged:

- Data-processing instructions without the S bit set

- LDR instructions

- LDM instructions, other than Load Multiple with Restore CPSR

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions write both the PC and PSR part of R15:

- Data-processing instructions with the S bit set

- Load Multiple with Restore CPSR

26-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions write both the PC and PSR part of R15:

- Data-processing instructions with the S bit set

- Load Multiple with Restore CPSR

32-bit : writing register 15

In 26-bit architecture the value of R15 is written in three different ways:

The following instructions write both the PC and PSR part of R15:

- Data-processing instructions with the S bit set

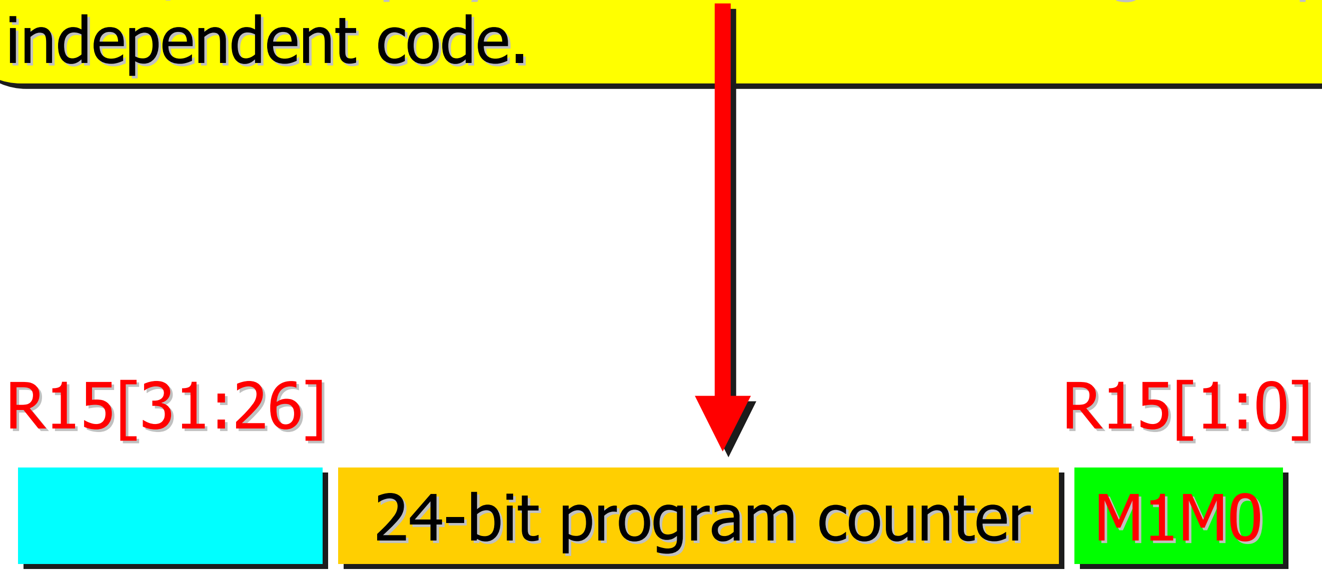
- Load Multiple with Restore CPSR

26-bit : writing PSP part in R15

Variants of the CMP, CMN, TST and TEQ instructions write just the PSR part of R15 and leave the PC part unchanged.

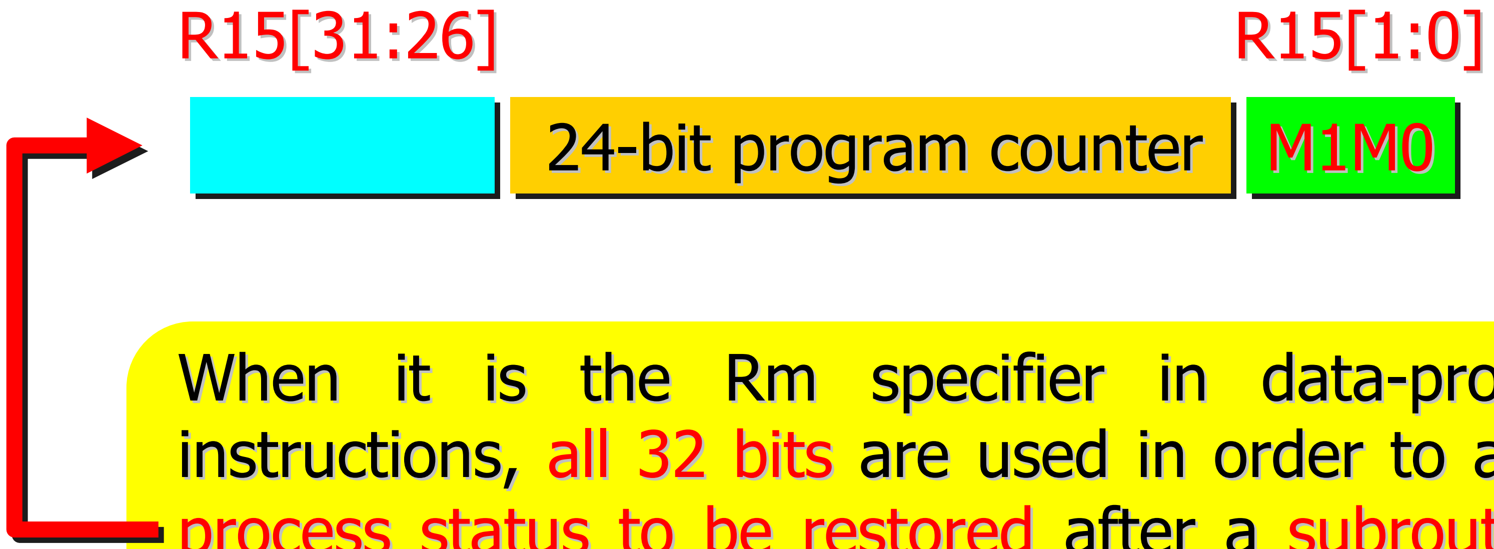
Register 15 : read/write rules

When it is the Rn specifier in data-processing instructions, or the base address for load and store instructions, only the value of the program counter is used , to simplify PC relative addressing and position-independent code.



P. Bakowski

Register 15 : read/write rules

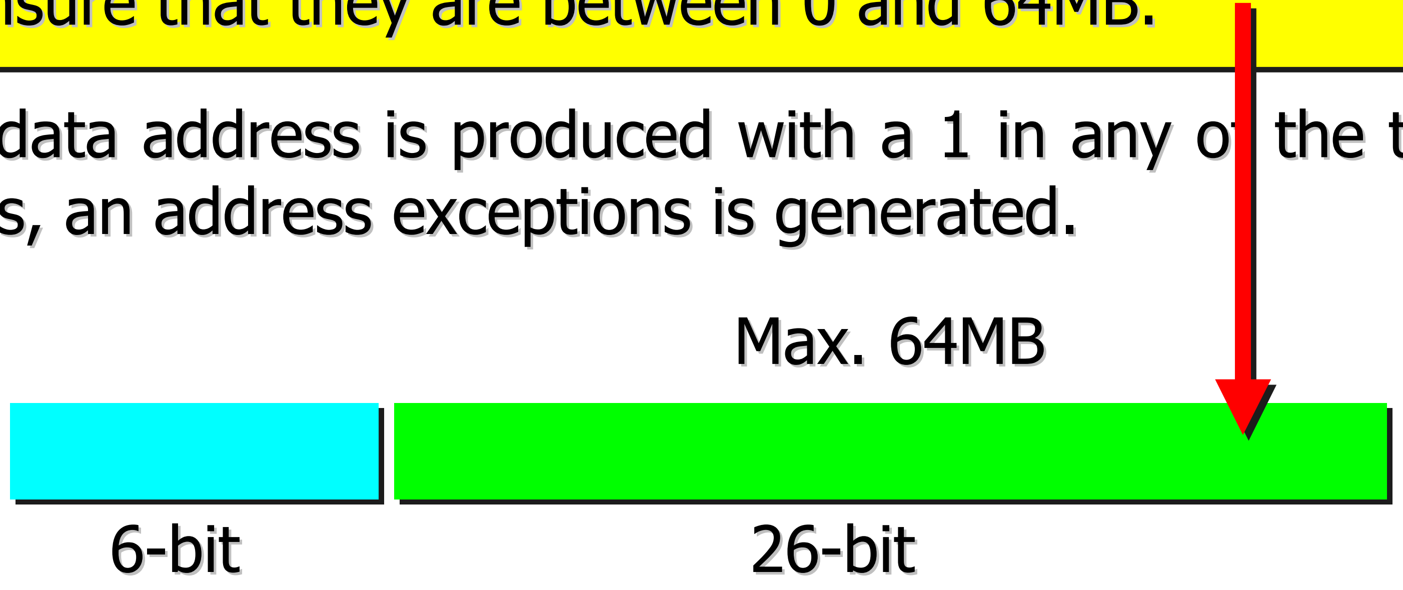


When it is the Rm specifier in data-processing instructions, all 32 bits are used in order to allow all process status to be restored after a subroutine call or exception by subroutine-return instructions such as: MOVS PC, LR and LDM ...

26-bit: address exception

In 26-bit architectures, all data addresses are checked to ensure that they are between 0 and 64MB.

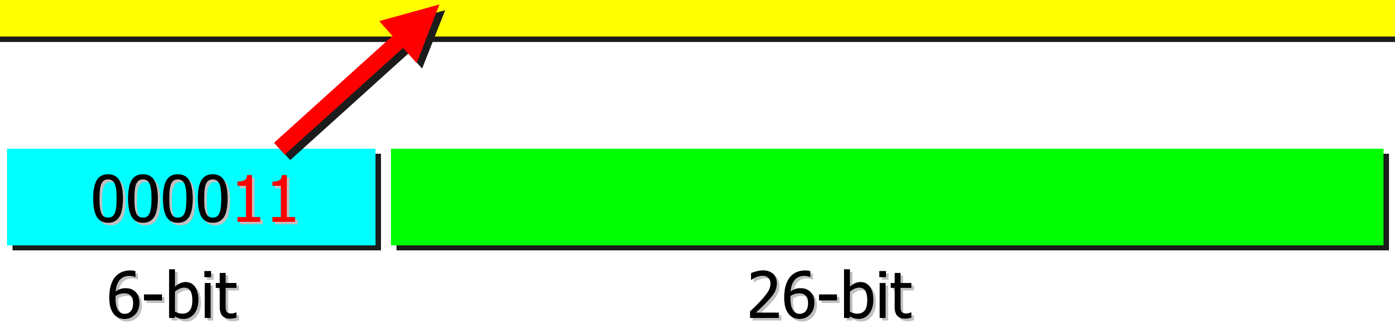
If a data address is produced with a 1 in any of the top 6 bits, an address exceptions is generated.



26-bit: address exception

In 26-bit architectures, all data addresses are checked to ensure that they are between 0 and 64MB.

If a data address is produced with a 1 in any of the top 6 bits, an address exceptions is generated.



26-bit: address exception

When an address exception is generated, the following actions are performed:

- R14_svc[25:2]) = address of instruction + 8
- R14_svc[31:26,1,0] = R15[31:26,1,0]
- M[1:0] = 0b11 ; supervisor mode
- F = unchanged
- I = 1 ; (normal) interrupts disabled
- PC = 0x14

The address of the instruction which caused the address exception is the value in R14 minus 8.

26-bit: address exception

When an address exception is generated, the following actions are performed:

- R14_svc[25:2]) = address of instruction + 8
- R14_svc[31:26,1,0] = R15[31:26,1,0]
- M[1:0] = 0b11 ; supervisor mode
- F = unchanged
- I = 1 ; (normal) interrupts disabled
- PC = 0x14

The address of the instruction which caused the address exception is the value in R14 minus 8.

26-bit: address exception

When an address exception is generated, the following actions are performed:

- R14_svc[25:2]) = address of instruction + 8
- R14_svc[31:26,1,0] = R15[31:26,1,0]
- M[1:0] = 0b11 ; supervisor mode
- F = unchanged
- I = 1 ; (normal) interrupts disabled
- PC = 0x14

The address of the instruction which caused the address exception is the value in R14 minus 8.

26-bit: address exception

When an address exception is generated, the following actions are performed:

- R14_svc[25:2]) = address of instruction + 8
- R14_svc[31:26,1,0] = R15[31:26,1,0]
- M[1:0] = 0b11 ; supervisor mode
- F = unchanged
- I = 1 ; (normal) interrupts disabled
- PC = 0x14

The address of the instruction which caused the address exception is the value in R14 minus 8.

26-bit: address exception

When an address exception is generated, the following actions are performed:

- R14_svc[25:2]) = address of instruction + 8
- R14_svc[31:26,1,0] = R15[31:26,1,0]
- M[1:0] = 0b11 ; supervisor mode
- F = unchanged
- I = 1 ; (normal) interrupts disabled
- PC = 0x14 - interruption routine address

The address of the instruction which caused the address exception is the value in R14 minus 8.

26-bit: returning from an exception

As this exception implies a programming error, it is not usual to return form address exceptions, but if a return is required, use:

```
SUBS PC,R14,#8
```

This restores both the PC and PSR (from R14_svc) and returns to the instruction that generated the address exception.

26-bit: returning from an exception

As this exception implies a programming error, it is not usual to return form address exceptions, but if a return is required, use:

```
SUBS PC,R14,#8
```

This restores both the PC and PSR (from R14_svc) and returns to the instruction that generated the address exception.

26-bit: returning from an exception

As this exception implies a programming error, it is not usual to return form address exceptions, but if a return is required, use:

```
SUBS PC,R14,#8
```

This restores both the PC and PSR (from R14_svc) and returns to the instruction that generated the address exception.

26-bit: branches

In 26-bit architectures, there are no restrictions on branching backwards past location 0x0000000 or forwards past location 0x3FFFFFF.

Such branches wrap around to the other end of the 26-bit address space, and so have a different target address than they would had in a 32-bit architecture.

As a result, the signed 24-bit word offset in the B and BL instructions allows any instruction in the 26-bit address space to be branched to.

26-bit: branches

In 26-bit architectures, there are no restrictions on branching backwards past location 0x0000000 or forwards past location 0x3FFFFFF.

Such branches wrap around to the other end of the 26-bit address space, and so have a different target address than they would had in a 32-bit architecture.

As a result, the signed 24-bit word offset in the B and BL instructions allows any instruction in the 26-bit address space to be branched to.

26-bit: branches

In 26-bit architectures, there are no restrictions on branching backwards past location 0x0000000 or forwards past location 0x3FFFFFF.

Such branches wrap around to the other end of the 26-bit address space, and so have a different target address than they would had in a 32-bit architecture.

As a result, the signed 24-bit word offset in the B and BL instructions allows any instruction in the 26-bit address space to be branched to.

26-bit versus 32-bit architectures

26-bit architectures

All process status (namely the condition flags, interrupt status and processor mode) can be preserved across subroutine calls and nested exceptions without adding any instructions to the entry or exit sequence.

26-bit versus 32-bit architectures

32-bit architectures

This process status functionality is given up to allow 32-bit instruction addresses to be used.

For exceptions, processor status is preserved in the SPSRs, and if nested exceptions using the same SPSR can occur, extra instructions are used to preserve this status in memory.

For subroutine calls, processor status can be preserved across the subroutine call by using extra instructions, but this is not normally done.

26-bit versus 32-bit architectures

32-bit architectures

This process status functionality is given up to allow 32-bit instruction addresses to be used.

For exceptions, processor status is preserved in the SPSRs, and if nested exceptions using the same SPSR can occur, extra instructions are used to preserve this status in memory.

For subroutine calls, processor status can be preserved across the subroutine call by using extra instructions, but this is not normally done.

26-bit versus 32-bit architectures

32-bit architectures

This process status functionality is given up to allow 32-bit instruction addresses to be used.

For exceptions, processor status is preserved in the SPSRs, and if nested exceptions using the same SPSR can occur, extra instructions are used to preserve this status in memory.

For subroutine calls, processor status can be preserved across the subroutine call by using extra instructions, but this is not normally done.

Summary

- ACORN , RISC 1 , and ARM
- ARM architecture versions
- ARMv1, ARMv2 and 26-bit addressing
- ARMv3 and 32-bit addressing
- Overview of the architectures

Summary

- ACORN , RISC 1 , and ARM
- ARM architecture versions
- ARMv1, ARMv2 and 26-bit addressing
- ARMv3 and 32-bit addressing
- Overview of the architectures

Summary

- ACORN , RISC 1 , and ARM
- ARM architecture versions
- ARMv1, ARMv2 and 26-bit addressing
- ARMv3 and 32-bit addressing
- Overview of the architectures

Summary

- ACORN , RISC 1 , and ARM
- ARM architecture versions
- ARMv1, ARMv2 and 26-bit addressing
- ARMv3 and 32-bit addressing
- Overview of the architectures

Summary

- ACORN , RISC 1 , and ARM
- ARM architecture versions
- ARMv1, ARMv2 and 26-bit addressing
- ARMv3 and 32-bit addressing
- Overview of the architectures