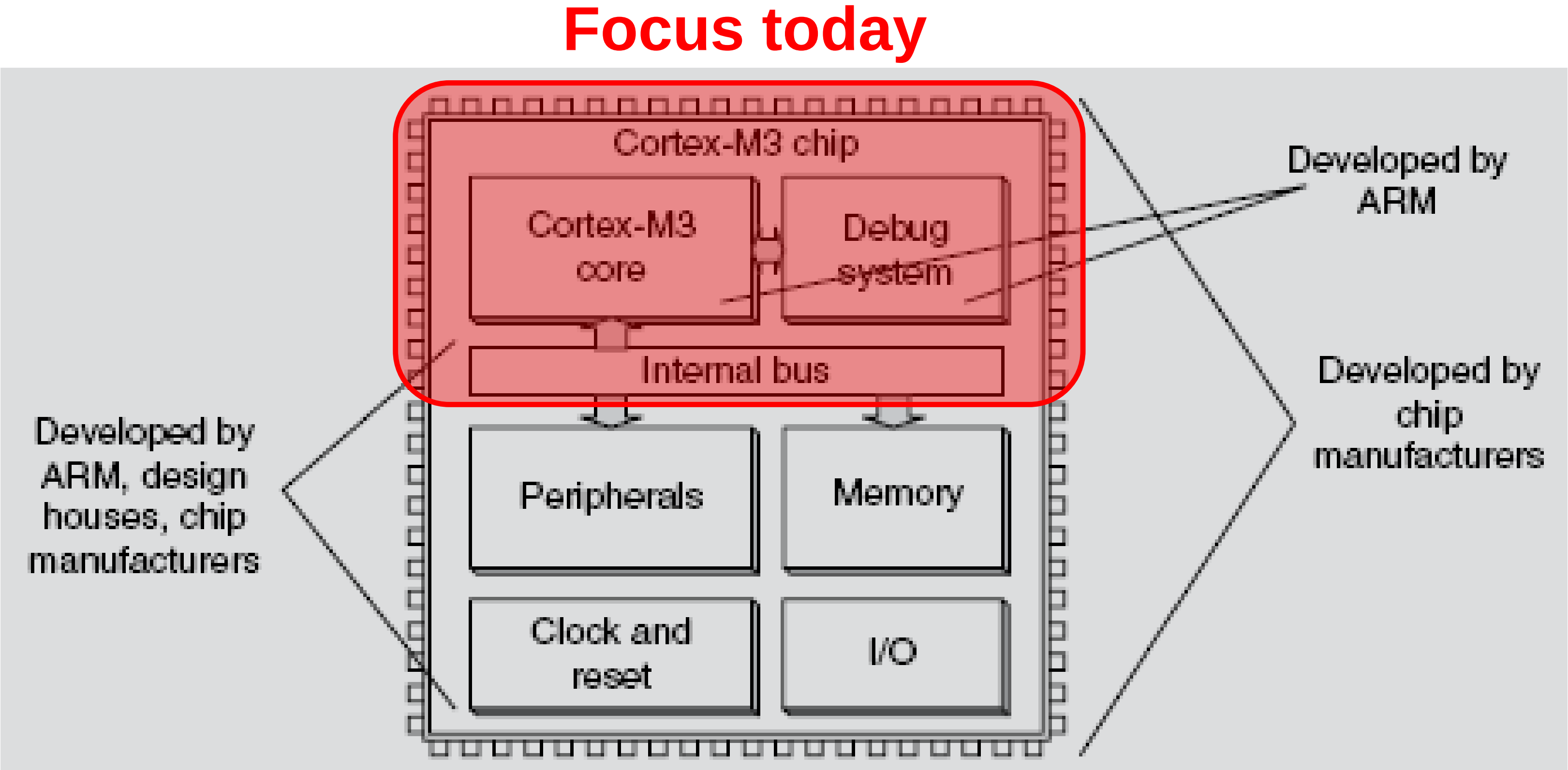
ARM Cortex-M3 Instruction Set & Architecture

Why another Micro

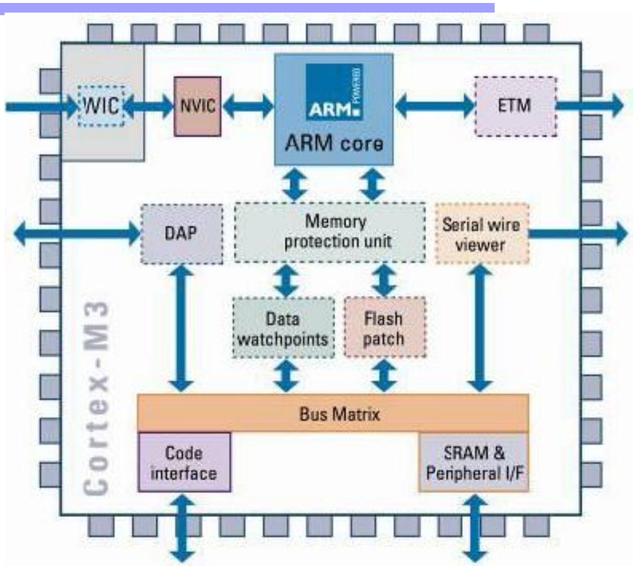
- *Greater performance efficiency*: allowing more work to be done without increasing the frequency or power requirements
- *Low power consumption*: enabling longer battery life, especially critical in portable products including wireless networking applications
- E*nhanced determinism*: guaranteeing that critical tasks and interrupts are serviced as quickly as possible and in a known number of cycles
- *Improved code density*: ensuring that code fits in a small memory footprint
- *Ease of use*: providing easier programmability and debugging for the growing number of 8-bit and 16-bit users migrating to 32 bits
- *Lower cost solutions*: reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit microcontrollers to be priced at less than US$1 for the first time
- *Wide choice of development tools*: from low-cost or free compilers to full-featured development suites from many development tool vendors

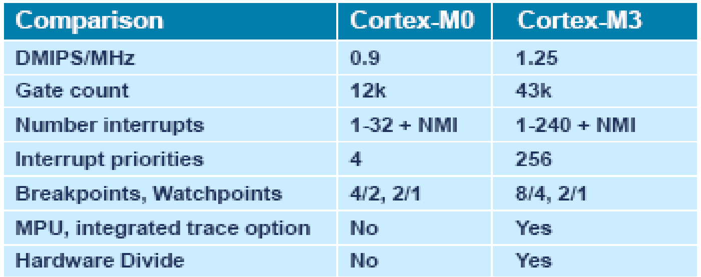
Processor vs. MCU



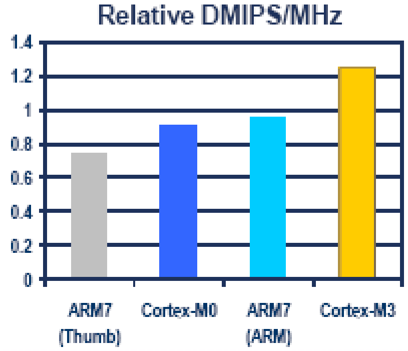
ARM Cortex-M3 Processor

- Cortex-M3 architecture
- Harvard bus architecture
  - 3-stage pipeline with branch speculation
- Configurable nested vectored interrupt controller (NVIC)
- Wake-up Interrupt Controller (WIC)
  - Enables ultra low-power standby operation
- Extended configurability of debug and trace capabilities
  - More flexibility for meeting specific market requirements
- Optional components for specific market reqs.
  - Memory Protection Unit (MPU)
  - Embedded Trace Macrocell™(ETM™)
- Support for fault robust implementations via configurable observation interface
  - EC61508 standard SIL3 certification
- Physical IP support
  - Power Management Kit™(PMK) + low-power standard cell libraries and memories enable0.18µm Ultra-Low Leakage (ULL) process





Comparison	Cortex-M0	Cortex-M3
DMIPS/MHz	0.9	1.25
Gate count	12k	43k
Number interrupts	1-32 + NMI	1-240 + NMI
Interrupt priorities	4	256
Breakpoints, Watchpoints	4/2, 2/1	8/4, 2/1
MPU, integrated trace option	No	Yes
Hardware Divide	No	Yes



ARM Architecture roadmap

4T

ARM7TDMI
ARM922T

Thumb instruction set

5TE

ARM926EJ-S
ARM946E-S
ARM966E-S

Improved ARM/Thumb Interworking

DSP instructions

Extensions:

Jazelle (5TEJ)

6

ARM1136JF-S
ARM1176JZF-S
ARM11 MPCore

SIMD Instructions

Unaligned data support

Extensions:

Thumb-2 (6T2)

TrustZone (6Z)

Multicore (6K)

7

Cortex-A8/R4/M3/M1

Thumb-2

Extensions:

v7A (applications) – NEON

v7R (real time) – HW Divide

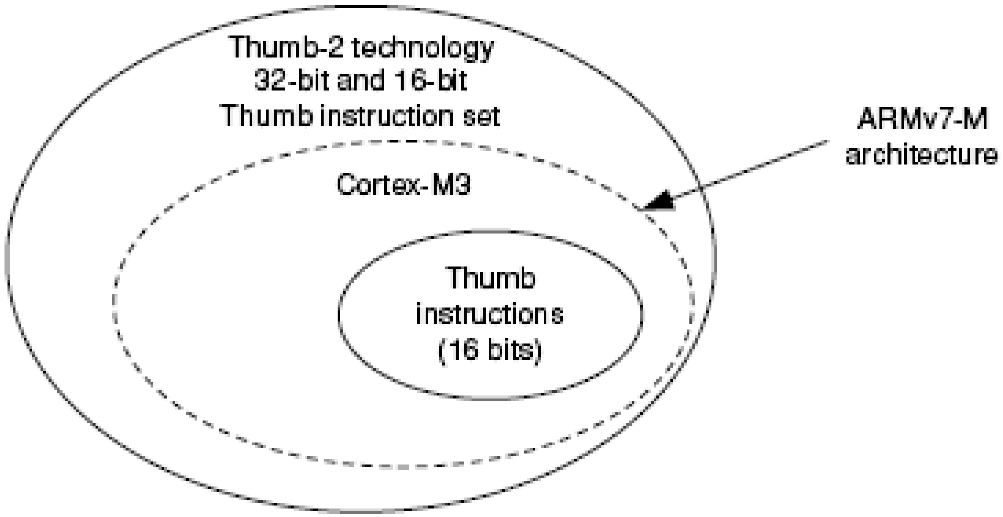
V7M (microcontroller) – HW Divide and Thumb-2 only

Which architecture is my processor?

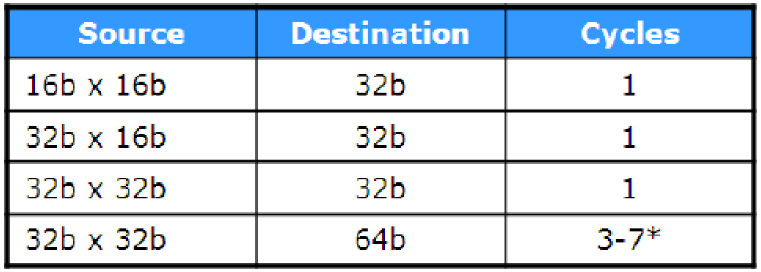
Processor core	Architecture
• ARM7TDMI family	v4T
– ARM720T, ARM740T	
• ARM9TDMI family	v4T
– ARM920T,ARM922T,ARM940T	
• ARM9E family	v5TE, v5TEJ
– ARM946E-S, ARM966E-S, ARM926EJ-S	
• ARM10E family	v5TE, v5TEJ
– ARM1020E, ARM1022E, ARM1026EJ-S	
• ARM11 family	v6
– ARM1136J(F)-S	
– ARM1156T2(F)-S	v6T2
– ARM1176JZ(F)-S	v6Z
• Cortex family	
– ARM Cortex-A8	v7A
– ARM Cortex-R4	v7R
– ARM Cortex-M3	v7M

Thumb-2

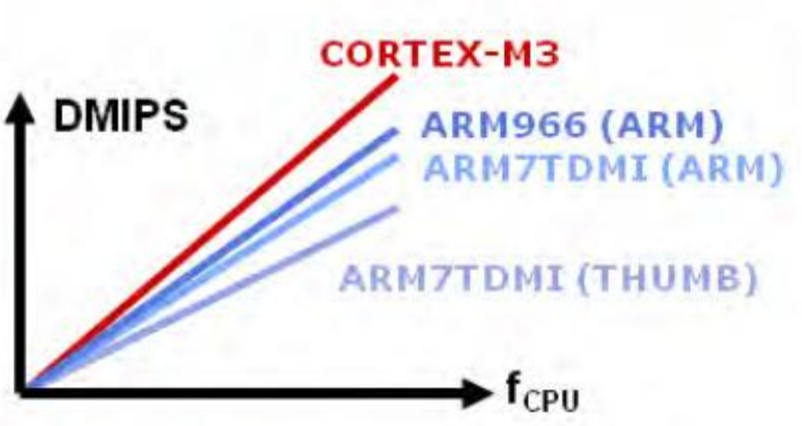
- Mixes 16 and 32 bits instructions
  - Enhancements: eg. UDIV, SDIF division, bit-field operators UFBX, BFC, BFE, wrt traditional ARMv4T
  - No need to mode switch, can be mixed freely
- **Not** backwards binary compatible
  - But porting is «easy»

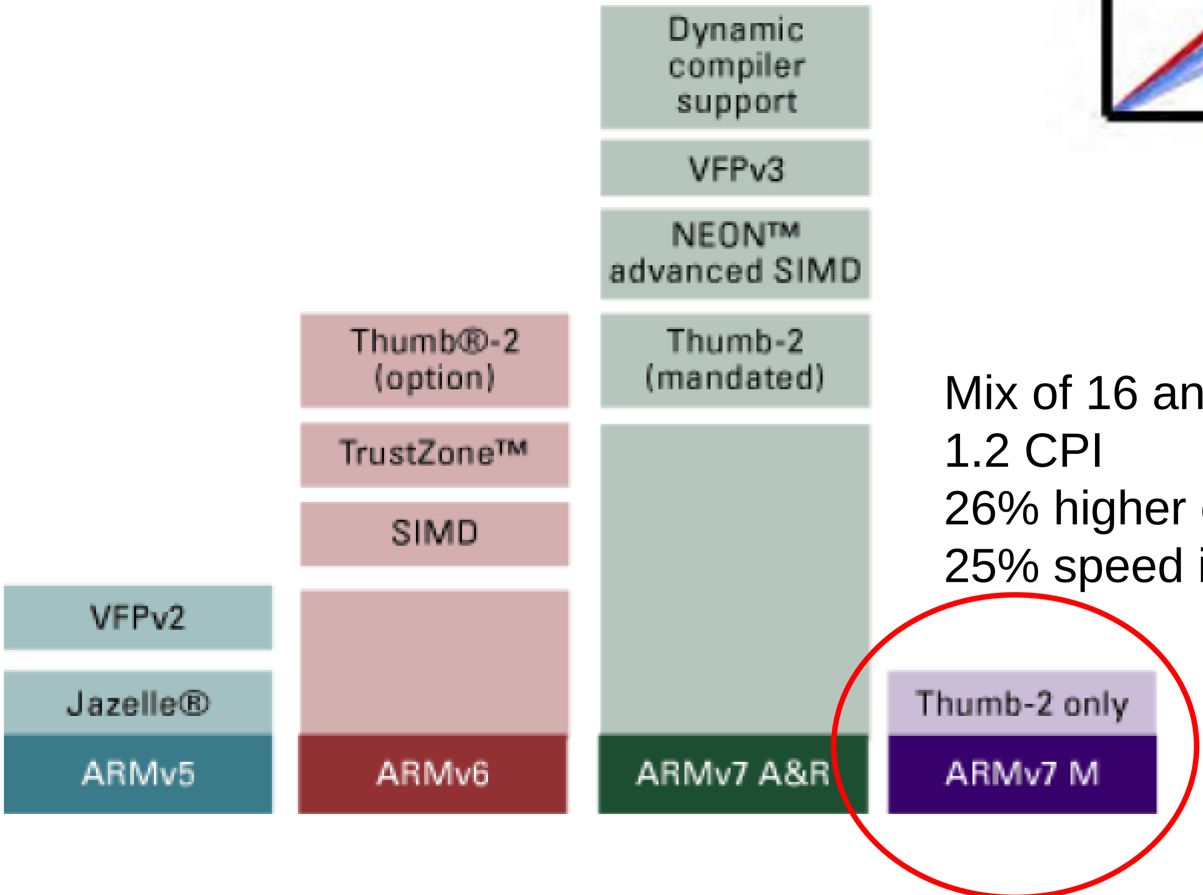


ARMv7 M (Thumb-2) features



Source	Destination	Cycles
16b x 16b	32b	1
32b x 16b	32b	1
32b x 32b	32b	1
32b x 32b	64b	3-7*





Dynamic compiler support

VFPv3

NEON™ advanced SIMD

Thumb-2 (mandated)

Thumb®-2 (option)

TrustZone™

SIMD

VFPv2

Jazelle®

Thumb-2 only

ARMv5

ARMv6

ARMv7 A&R

ARMv7 M

Mix of 16 and 32b instructions
1.2 CPI
26% higher code density ARM32
25% speed improvement over Thumb16

Cortex-M3 features

Low-gate count with advanced features

- ARMv7-M: A Thumb-2 ISA subset, consisting of all base Thumb-2 instructions, 16-bit and 32-bit, and excluding blocks for media, SIMD, E (DSP), and ARM system access.
- Banked SP only
- Hardware divide instructions, SDIV and UDIV (Thumb-2 instructions)
- Handler and Thread modes
- Thumb and Debug states.
- Interruptible-continued LDM/STM, PUSH/POP for low interrupt latency.
- Automatic processor state saving and restoration for low latency Interrupt Service Routine (ISR) entry and exit.
- ARM architecture v6 style BE8/LE support.
- ARMv6 unaligned accesses.

M3 - Enhanced Interrupt support

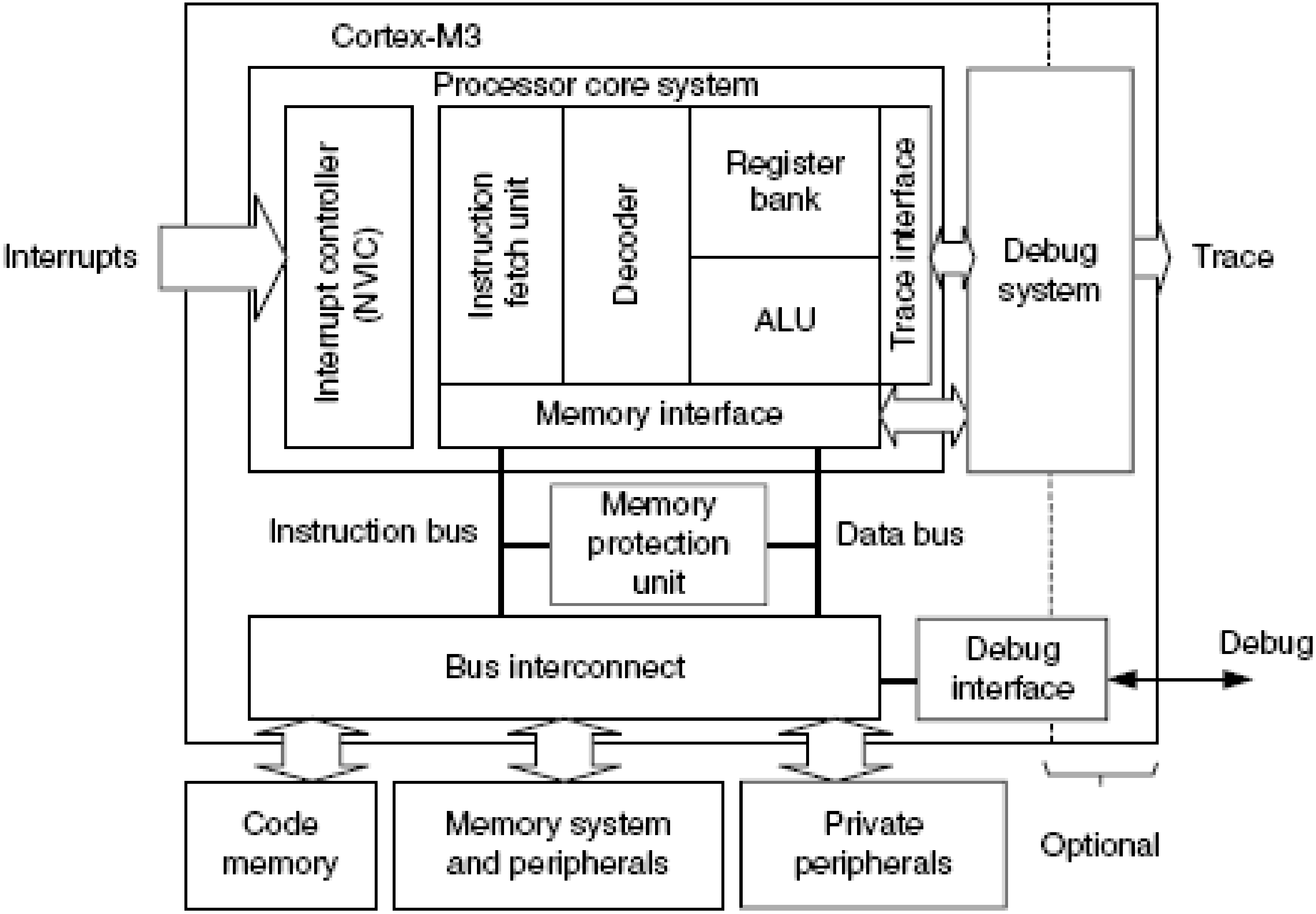
Nested Vectored Interrupt Controller (NVIC) integrated with the processor for low latency

- Configurable number, 1 to 240, of external interrupts
- Configurable number, 3 to 8, of bits of priority.
- Dynamic reprioritization of interrupts.
- Priority grouping. This allows selection of pre-empting interrupt levels and non pre-empting interrupt levels
- Support for tail-chaining, and late arrival, of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts
- Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.

Memory, Peripheral, Debug IFs

- Optional Memory Protection Unit (MPU)
  - –Eight memory regions.
  - –Sub Region Disable (SRD), enabling efficient use of memory regions.
  - –Background region can be enabled which implements the default memory map attributes.
- Bus interfaces:
  - –AHBLite ICode, DCode and System bus interfaces.
  - –APB Private Peripheral Bus (PPB) Interface
  - –Bit band support. Atomic bit-band write and read operations.
  - –Memory access alignment.
  - –Write buffer. For buffering of write data.
- Low-cost debug solution:
  - –Debug access to all memory and registers in the system, including Cortex-M3 register bank when the core is running, halted, or held in reset.
  - –Serial Wire (SW-DP) or JTAG (JTAG-DP) debug access, or both.
  - –Flash Patch and Breakpoint unit (FPB) for implementing breakpoints and code patches.
  - –Data Watchpoint and Trigger unit (DWT) for implementing watchpoints, trigger resources, and system profiling.
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging.
  - –Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer.
  - –Optional Embedded Trace Macrocell (ETM) for instruction trace.

Architecture Diagram



Pipeline

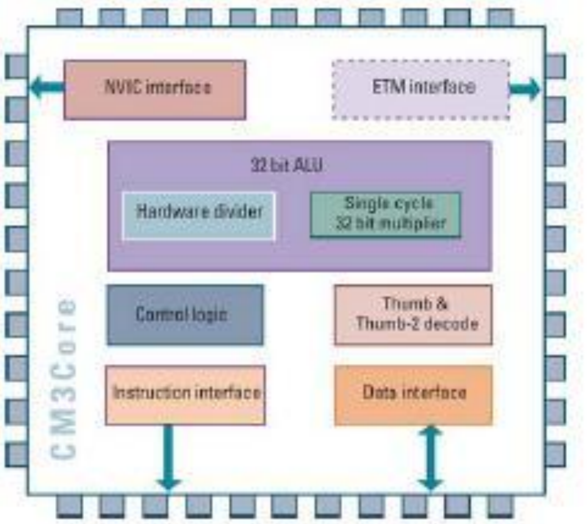
Harvard architecture

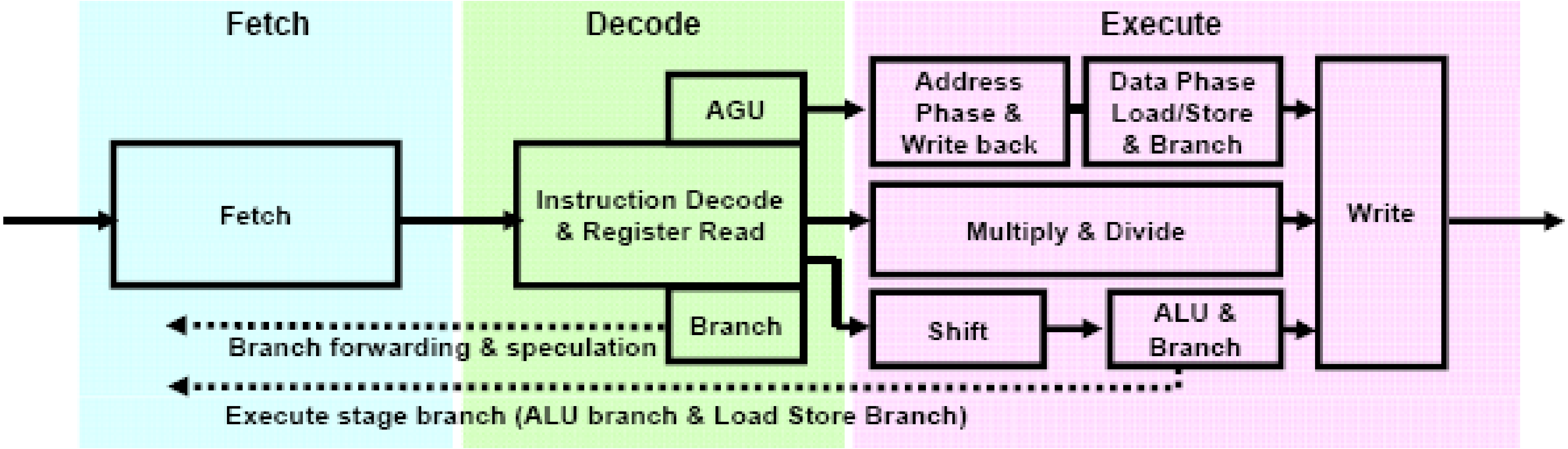
- Separate Instruction & Data buses enable parallel fetch & store

Advanced 3-Stage Pipeline

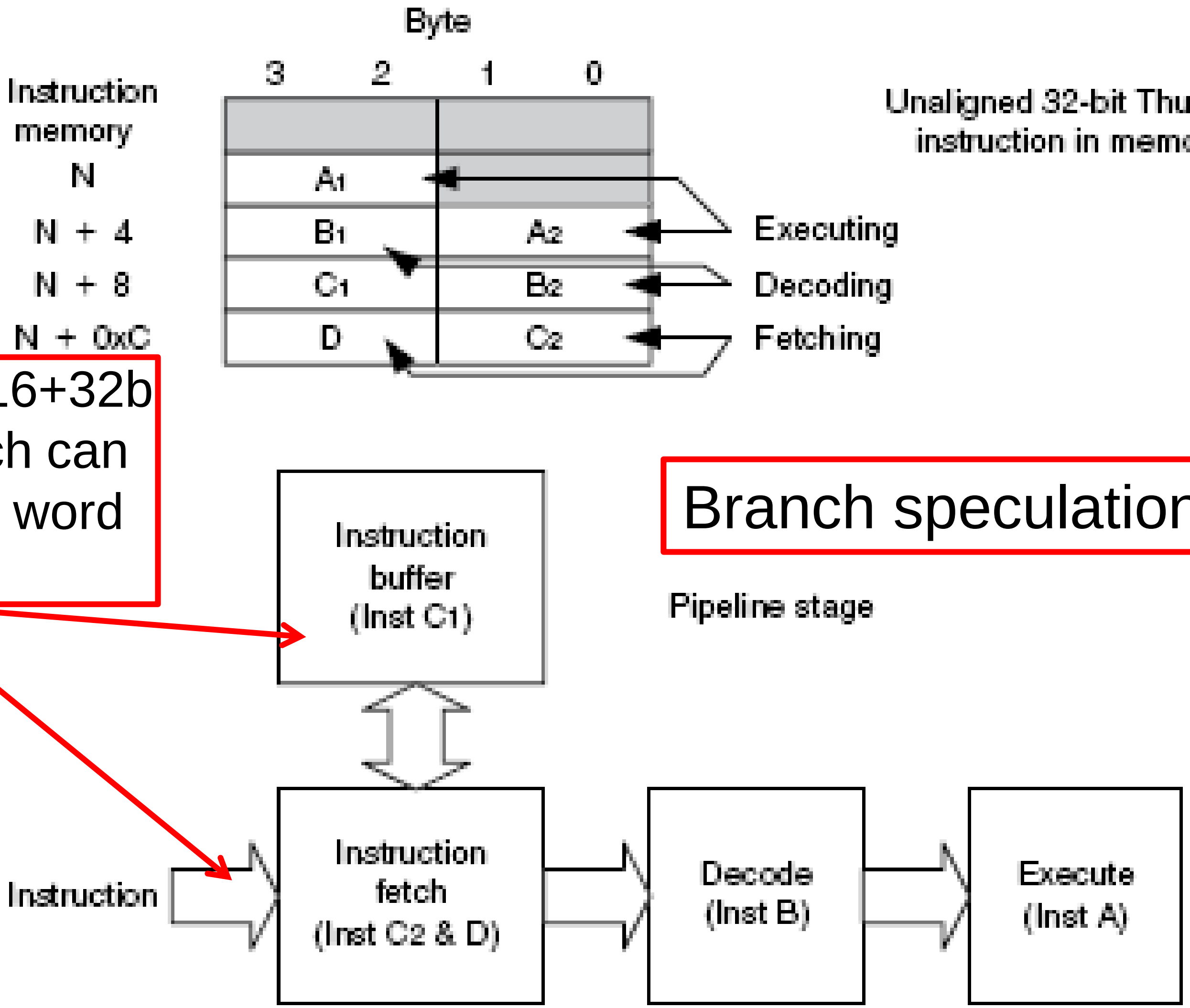
- Includes Branch Forwarding & Speculation

Additional Write-Back via Bus Matrix





Instruction Prefetch & Execution



Handles mix of 16+32b instructions which can be misaligned in word address

Branch speculation

Processor Modes

- The ARM has seven basic operating modes:
  - Each mode has access to:
    - Its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

Exception modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	**Privileged modes**
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	**Unprivileged mode**

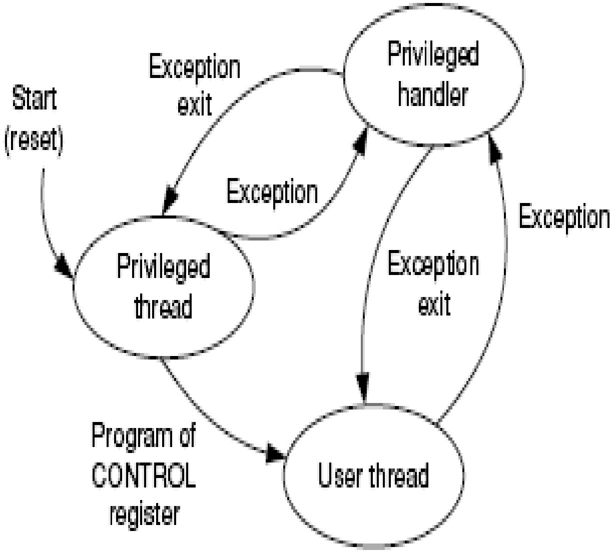
Operating Modes

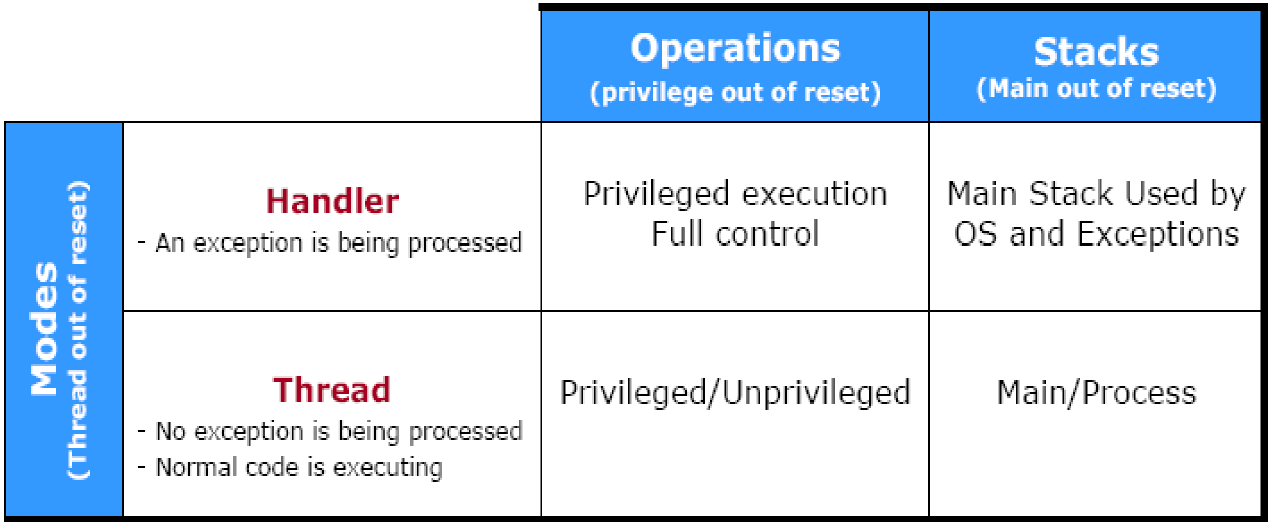
User mode:

- Normal program execution mode
- System resources unavailable
- Mode changed by exception only

Exception modes:

- Entered upon exception
- Full access to system resources
- Mode changed freely





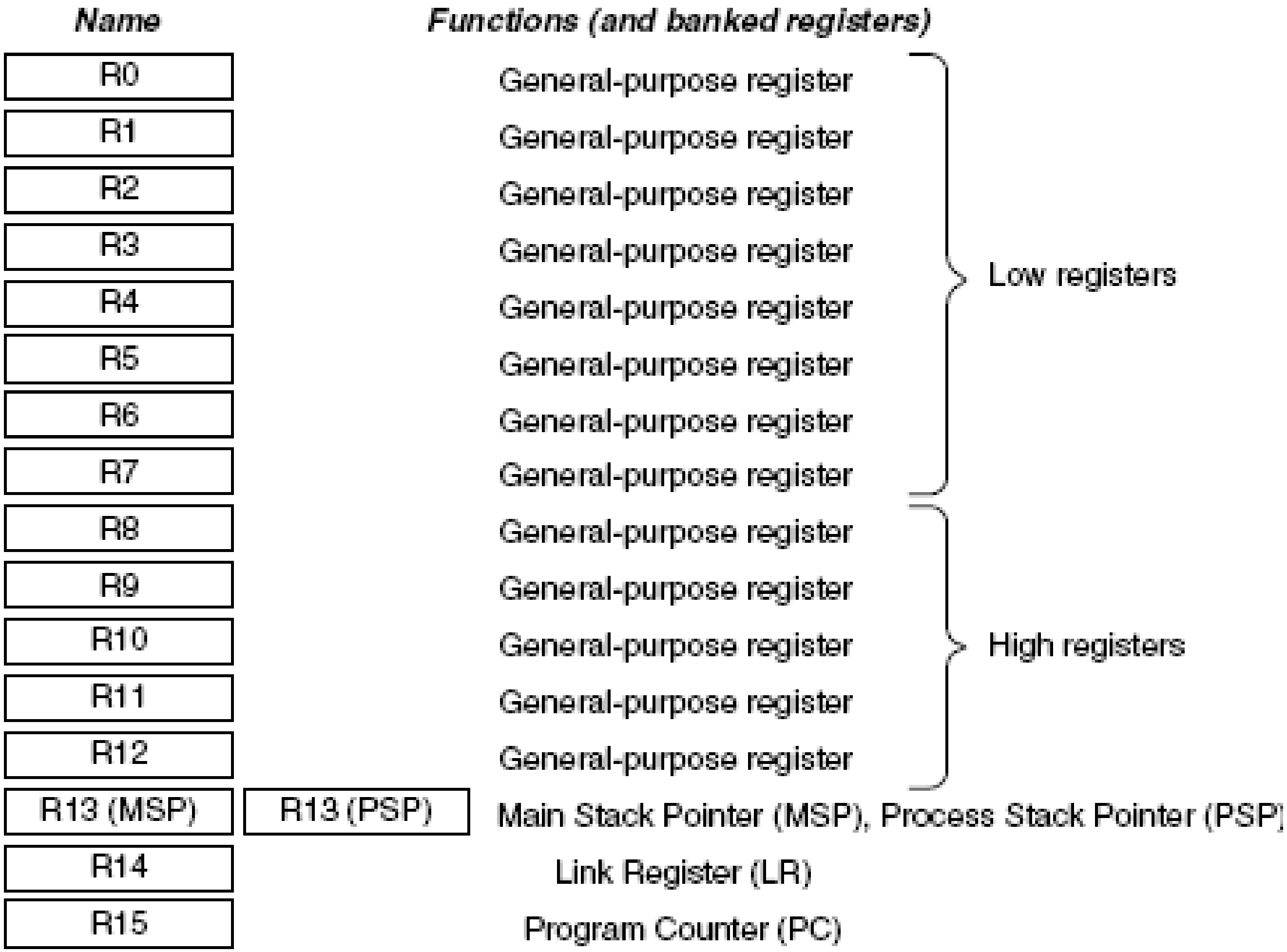
		Operations (privilege out of reset)	Stacks (Main out of reset)
Modes (Thread out of reset)	**Handler** - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

Exceptions

Exception	Mode	Priority	IV Address
Reset	Supervisor	1	0x00000000
Undefined instruction	Undefined	6	0x00000004
Software interrupt	Supervisor	6	0x00000008
Prefetch Abort	Abort	5	0x0000000C
Data Abort	Abort	2	0x00000010
Interrupt	IRQ	4	0x00000018
Fast interrupt	FIQ	3	0x0000001C

Table 1 - Exception types, sorted by Interrupt Vector addresses

Registers



Name		*Functions (and banked registers)*	
R0		General-purpose register	Low registers
R1		General-purpose register	
R2		General-purpose register	
R3		General-purpose register	
R4		General-purpose register	
R5		General-purpose register	
R6		General-purpose register	
R7		General-purpose register	
R8		General-purpose register	High registers
R9		General-purpose register	
R10		General-purpose register	
R11		General-purpose register	
R12		General-purpose register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	

ARM Registers

- 31 general-purpose 32-bit registers
- 16 visible, R0 – R15
- Others speed up the exception process

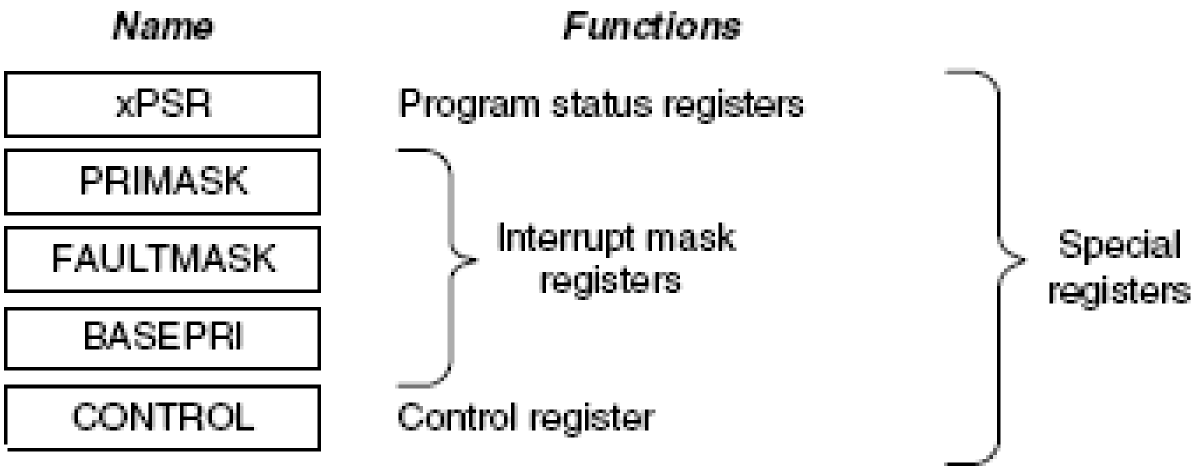
ARM Registers (2)

- Special roles:
  - Hardware
    - R14 – Link Register (LR): optionally holds return address for branch instructions
    - R15 – Program Counter (PC)
  - Software
    - R13 - Stack Pointer (SP)

ARM Registers (3)

- Current Program Status Register (CPSR)
- Saved Program Status Register (SPSR)
- On exception, entering *mod* mode:
  - (PC + 4) → LR
  - CPSR → SPSR_mod
  - PC ← IV address
  - R13, R14 replaced by R13_mod, R14_mod
  - In case of FIQ mode R7 – R12 also replaced

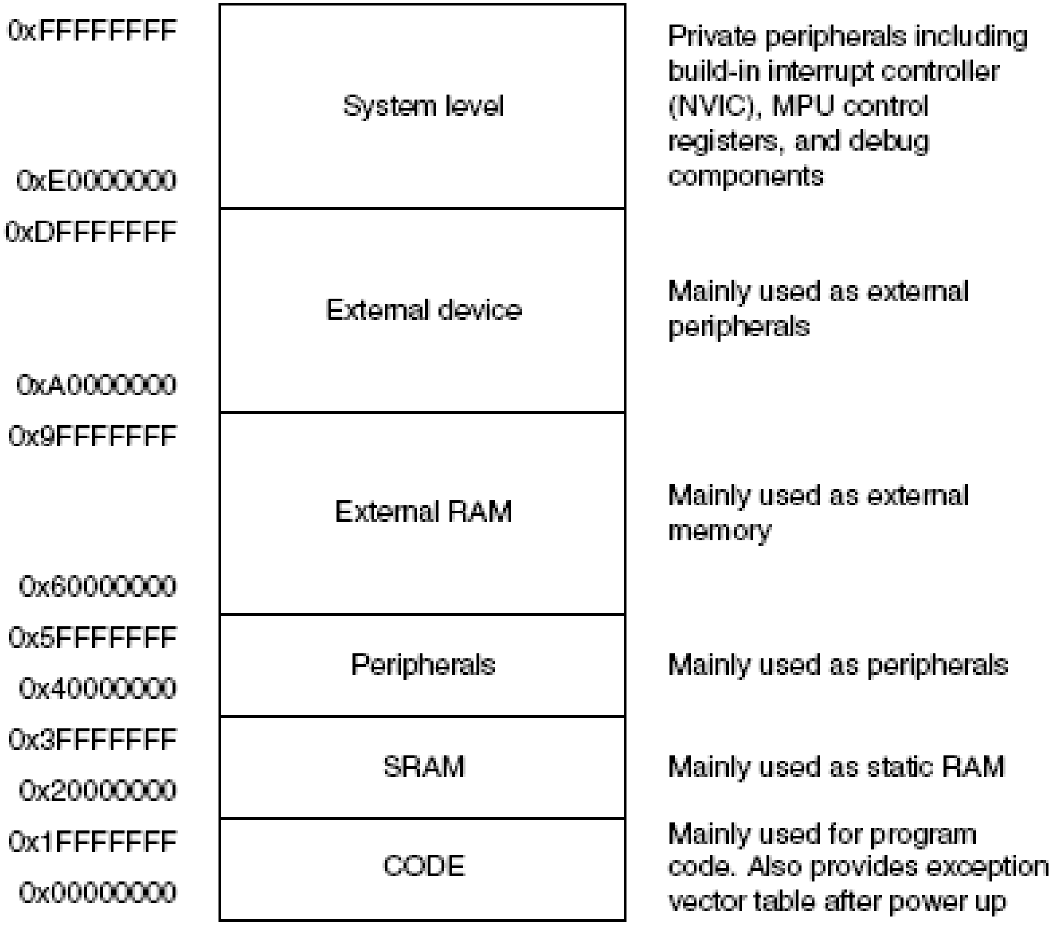
Special Registers



Register	Function
xPSR	Provide arithmetic and logic processing flags (zero flag and carry flag), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and hard fault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

Memory map

- Statically defined memory map (faster addr decoding) 4GB of address psace



0xFFFFFFFF

System level

Private peripherals including build-in interrupt controller (NVIC), MPU control registers, and debug components

0xE0000000

0xDFFFFFFF

External device

Mainly used as external peripherals

0xA0000000

0x9FFFFFFF

External RAM

Mainly used as external memory

0x60000000

0x5FFFFFFF

Peripherals

Mainly used as peripherals

0x40000000

0x3FFFFFFF

SRAM

Mainly used as static RAM

0x20000000

0x1FFFFFFF

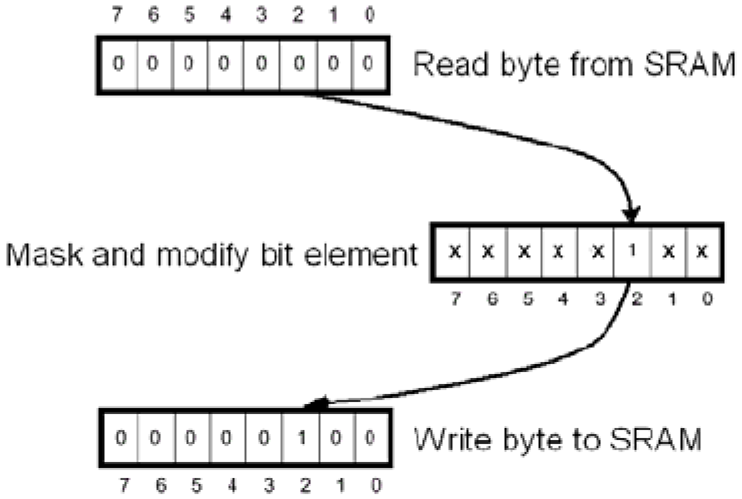
CODE

Mainly used for program code. Also provides exception vector table after power up

0x00000000

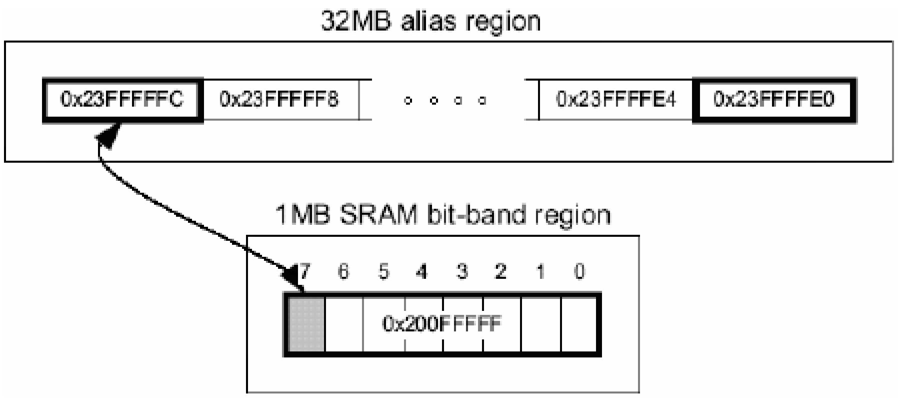
Bit Banding

- Fast single-bit manipulation: 1MB → 32MB aliased regions in SRAM & Peripheral space



```
LDR R0,=0x200FFFFF ; Setup address
MOV R2, #0x4       ; Setup data
LDR R1, [R0]       ; Read
ORR R1, R2         ; Modify bit
STR R1, [R0]       ; Write back result
```

Traditional bit manipulation method



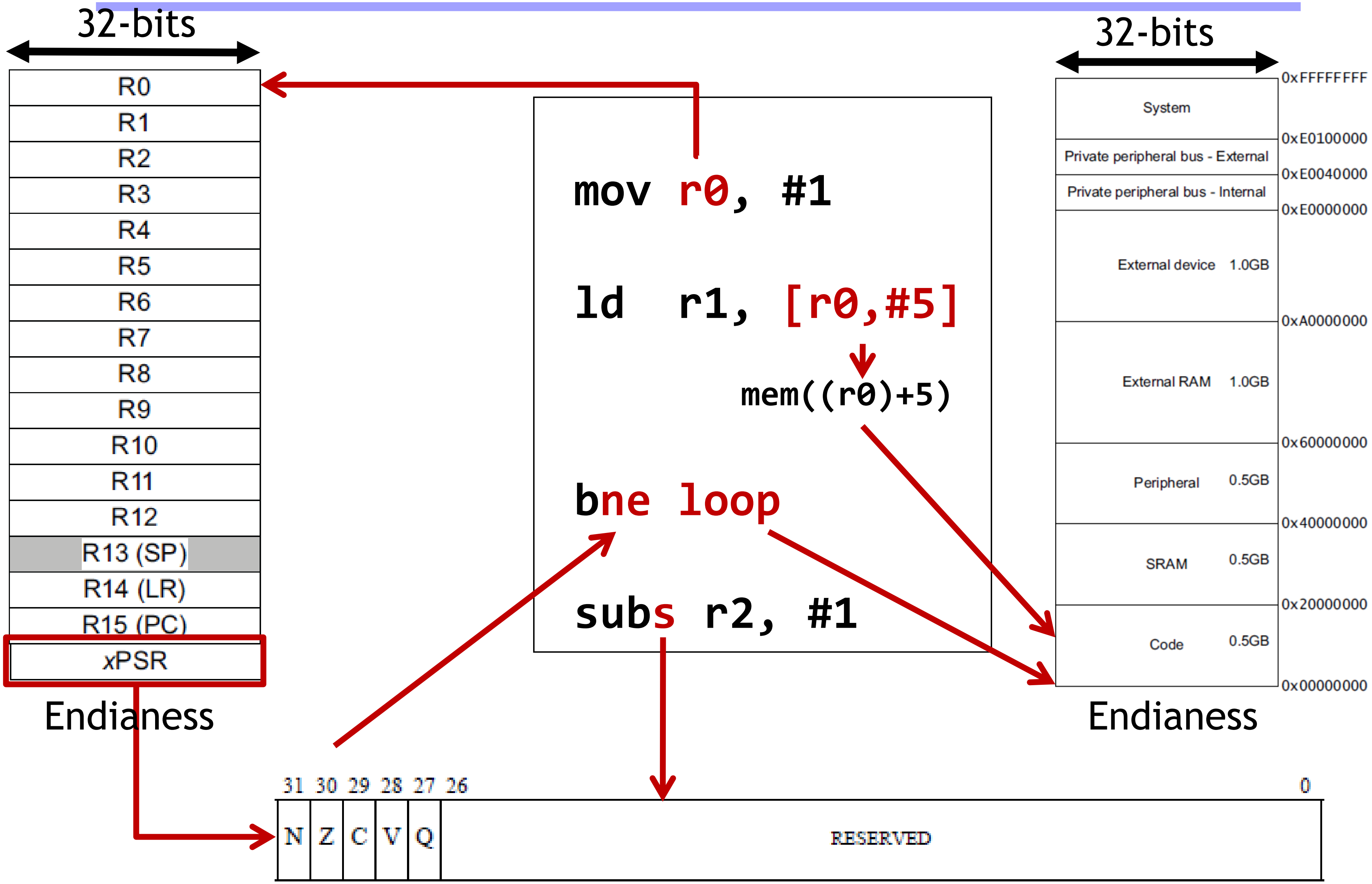
```
LDR R0,=0x23FFFFFC ; Setup address
MOV R1, #0x1       ; Setup data
STR R1, [R0]       ; Write
```

Direct, single cycle access with bit banding

Cortex M3 Instruction Set

Major Elements of ISA

(registers, memory, word size, endianess, conditions, instructions, addressing modes)



32-bits

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SP)
R14 (LR)
R15 (PC)
*x*PSR

Endianess

```
mov r0, #1

ld  r1, [r0,#5]
            mem((r0)+5)

bne loop

subs r2, #1
```

32-bits

Region	Size	Start address
System		0xE0100000 – 0xFFFFFFFF
Private peripheral bus - External		0xE0040000
Private peripheral bus - Internal		0xE0000000
External device	1.0GB	0xA0000000
External RAM	1.0GB	0x60000000
Peripheral	0.5GB	0x40000000
SRAM	0.5GB	0x20000000
Code	0.5GB	0x00000000

Endianess

31	30	29	28	27	26 … 0
N	Z	C	V	Q	RESERVED

Traditional ARM instructions

- Fixed length of 32 bits
- Commonly take two or three operands
- Process data held in registers
- Shift & ALU operation in single clock cycle
- Access memory with load and store instructions only
  - Load/Store multiple register
- Can be extended to execute conditionally by adding the appropriate suffix
- Affect the CPSR status flags by adding the ‘S’ suffix to the instruction

Thumb-2

- Original 16-bit Thumb instruction set
  - a subset of the full ARM instructions
  - performs similar functions to selective 32-bit ARM instructions but in 16-bit code size
- For ARM instructions that are not available
  - more 16-bit Thumb instructions are needed to execute the same function compared to using ARM instructions
  - but performance may be degraded
- Hence the introduction of the Thumb-2 instruction set
  - enhances the 16-bit Thumb instructions with additional 32-bit instructions
- All ARMv7 chips support the Thumb-2 (& ARM) instruction set
  - but Cortex-M3 supports only the 16-bit/32-bit Thumb-2 instruction set

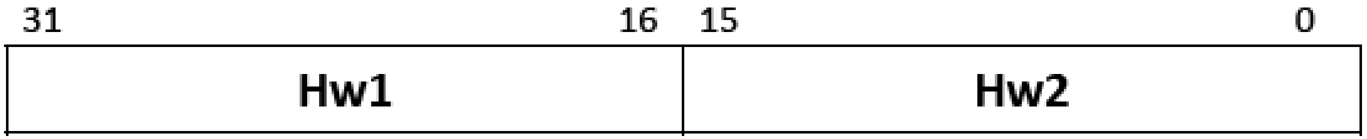
16bit Thumb-2

Some of the changes used to reduce the length of the instructions from 32 bits to 16 bits:

- reduce the number of bits used to identify the register
  - less number of registers can be used
- reduce the number of bits used for the immediate value
  - smaller number range
- remove options such as ‘S’
  - make it default for some instructions
- remove conditional fields (N, Z, V, C)
- no conditional executions (except branch)
- remove the optional shift (and no barrel shifter operation
  - introduce dedicated shift instructions
- remove some of the instructions
  - more restricted coding

Thumb-2 Implementation

- The 32-bit ARM Thumb-2 instructions are added through the space occupied by the Thumb BL and BLX instructions



32-bit Thumb-2 Instruction format

- The first Halfword (Hw1)
  – determines the instruction length and functionality
- If the processor decodes the instruction as 32-bit long
  – the processor fetches the second halfword (hw2) of the instruction from the instruction address plus two

Unified Assembly Language

- UAL supports generation of either Thumb-2 or ARM instructions from the same source code
  - same syntax for both the Thumb code and ARM code
  - enable portability of code for different ARM processor families
- Interpretation of code type is based on the directive listed in the assembly file
- Example:
  - For GNU GAS, the directive for UAL is

.syntax unified

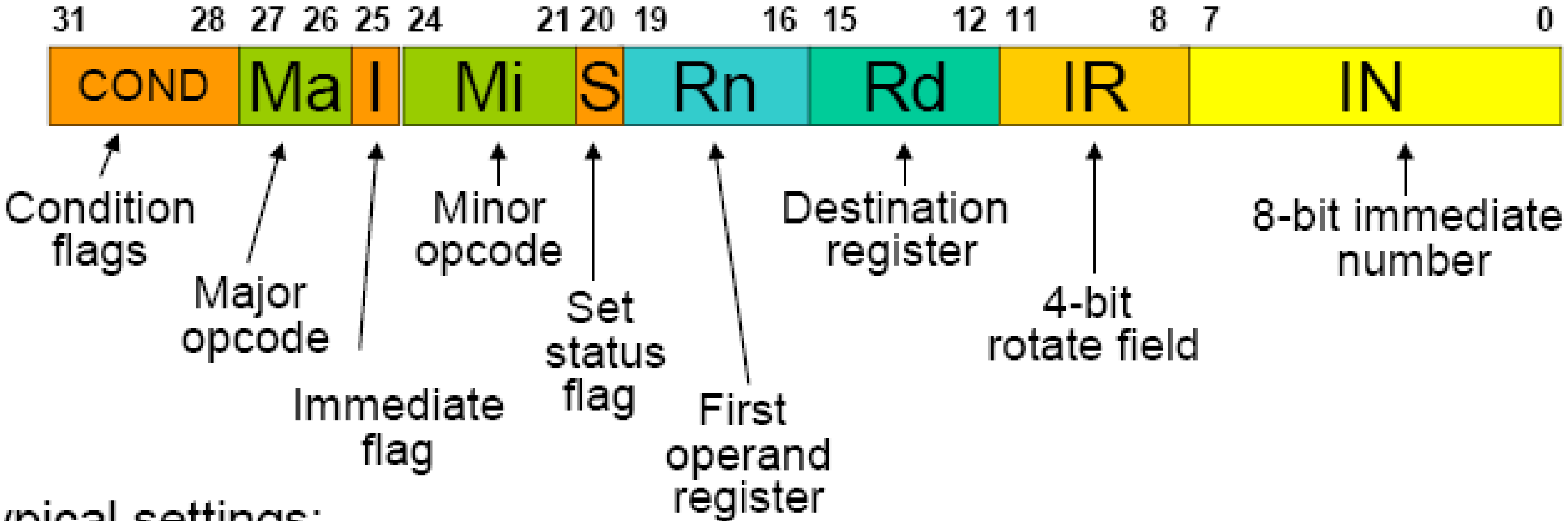
- For ARM assembler, the directive for UAL is

THUMB

32bit Instruction Encoding

Example: ADD instruction format

- ARM 32-bit encoding for ADD with immediate field



Typical settings:

Major opcode = 00 (this indicates data operation instructions)

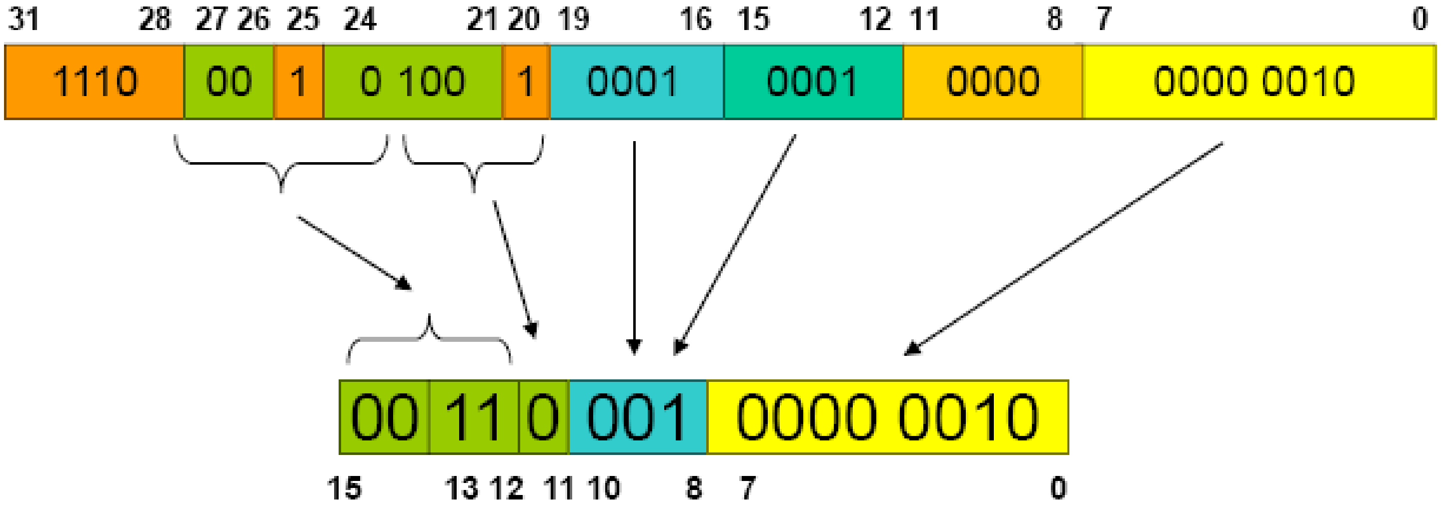
Minor opcode = 0100 (specifically, 100 ⇨ ADD instruction)

Immediate flag = 1 (immediate field in operand 2)

Set status flag = 1 (set carry flag after operation)

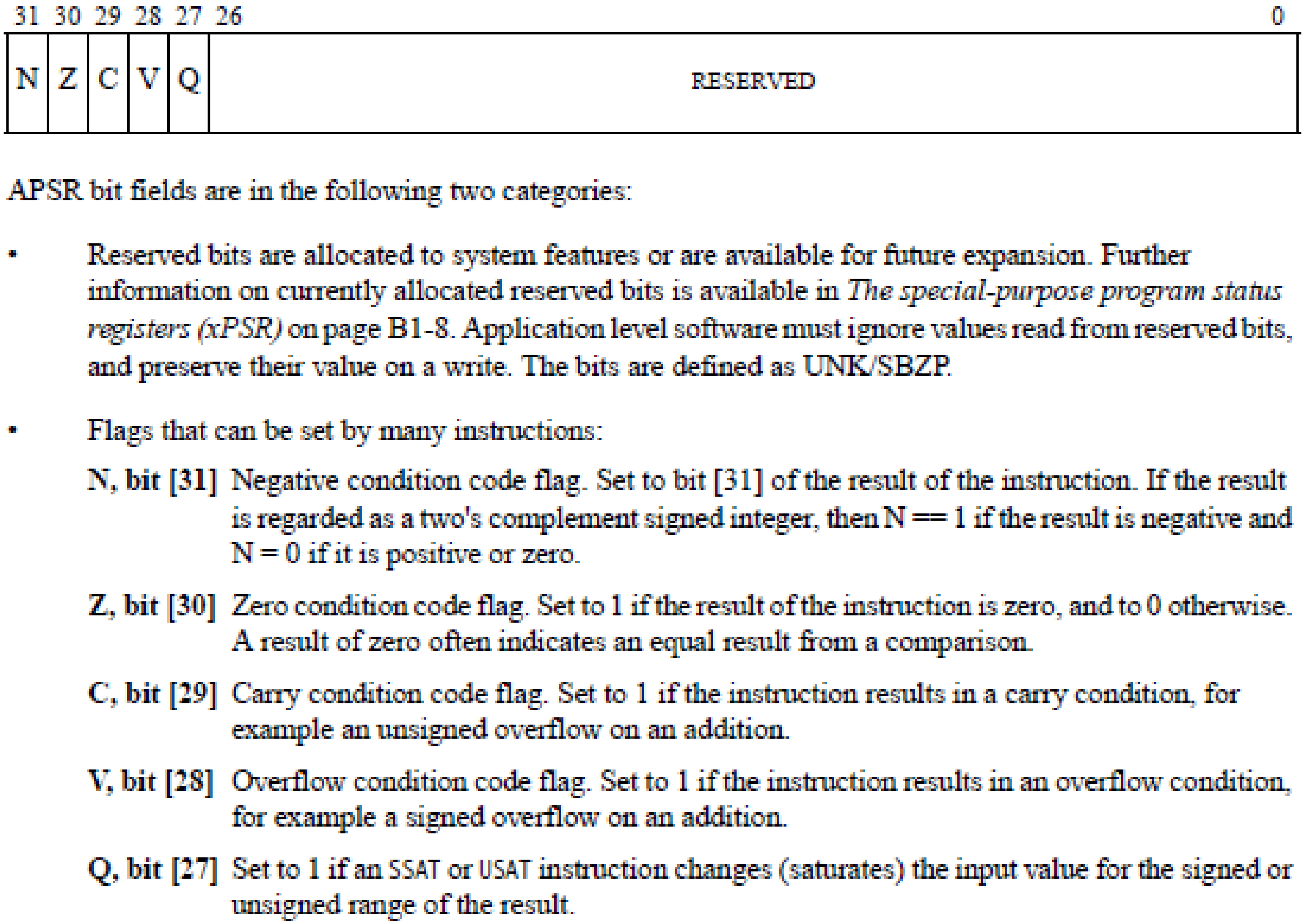
ARM and 16-bit Instruction Encoding

ARM 32-bit encoding: `ADDS r1, r1, #2`



- Equivalent 16-bit Thumb instruction: `ADD r1, #2`
  - No condition flag
  - No rotate field for the immediate number
  - Use 3-bit encoding for the register
  - Shorter opcode with implicit flag settings (e.g. the set status flag is always set)

Application Program Status Register (APSR)



31	30	29	28	27	26	0
N	Z	C	V	Q	RESERVED	

APSR bit fields are in the following two categories:

- Reserved bits are allocated to system features or are available for future expansion. Further information on currently allocated reserved bits is available in *The special-purpose program status registers (xPSR)* on page B1-8. Application level software must ignore values read from reserved bits, and preserve their value on a write. The bits are defined as UNK/SBZP.

- Flags that can be set by many instructions:

  N, bit [31] Negative condition code flag. Set to bit [31] of the result of the instruction. If the result is regarded as a two's complement signed integer, then N == 1 if the result is negative and N = 0 if it is positive or zero.

  Z, bit [30] Zero condition code flag. Set to 1 if the result of the instruction is zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

  C, bit [29] Carry condition code flag. Set to 1 if the instruction results in a carry condition, for example an unsigned overflow on an addition.

  V, bit [28] Overflow condition code flag. Set to 1 if the instruction results in an overflow condition, for example a signed overflow on an addition.

  Q, bit [27] Set to 1 if an SSAT or USAT instruction changes (saturates) the input value for the signed or unsigned range of the result.

Updating the APSR

- SUB Rx, Ry
  - Rx = Rx - Ry
  - APSR unchanged
- SUBS
  - Rx = Rx - Ry
  - APSR N or Z bits might be set
- ADD Rx, Ry
  - Rx = Rx + Ry
  - APSR unchanged
- ADDS
  - Rx = Rx + Ry
  - APSR C or V bits might be set

Conditional Execution

- Each data processing instruction prefixed by condition code
- Result – smooth flow of instructions through pipeline
- 16 condition codes:

EQ	equal	MI	negative	HI	unsigned higher	GT	signed greater than
NE	not equal	PL	positive or zero	LS	unsigned lower or same	LE	signed less than or equal
CS	unsigned higher or same	VS	overflow	GE	signed greater than or equal	AL	always
CC	unsigned lower	VC	no overflow	LT	signed less than	NV	special purpose

Conditional Execution

- Every ARM (32 bit) instruction is conditionally executed.
- The top four bits are ANDed with the CPSR condition codes, If they do not matched the instruction is executed as NOP
- The AL condition is used to execute the instruction irrespective of the value of the condition code flags.
- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". Ex: SUBS r1,r1,#1
- Conditional Execution improves code density and performance by reducing the number of forward branch instructions.

```
Normal                      Conditional
  CMP   r3,#0                       CMP   r3,#0
  BEQ   skip                        ADDNE r0,r1,r2
  ADD   r0,r1,r2
skip
```

Conditional Execution and Flags

- ARM instructions can be made to execute conditionally by post-fixing them with the appropriate condition code
  - This can increase code density and increase performance by reducing the number of forward branches

```
CMP     r0, r1
ADDGT   r2, r2, #1
ADDLE   r3, r3, #1
```

r0 - r1, compare r0 with r1 and set flags

if > r2=r2+1 flags remain unchanged

if <= r3=r3+1 flags remain unchanged

- By default, data processing instructions do not affect the condition flags but this can be achieved by post fixing the instruction (and any condition code) with an “S”

```
loop
    ADD     r2, r2, r3
    SUBS    r1, r1, #0x01
    BNE     loop
```

r2=r2+r3

decrement r1 and set flags

if Z flag clear then branch

Conditional execution examples

C source code

```
if (r0 == 0)
{
  r1 = r1 + 1;
}
else
{
  r2 = r2 + 1;
}
```

ARM instructions

unconditional

```
  CMP r0, #0
  BNE else
  ADD r1, r1, #1
  B end
else
  ADD r2, r2, #1
end
  ...
```

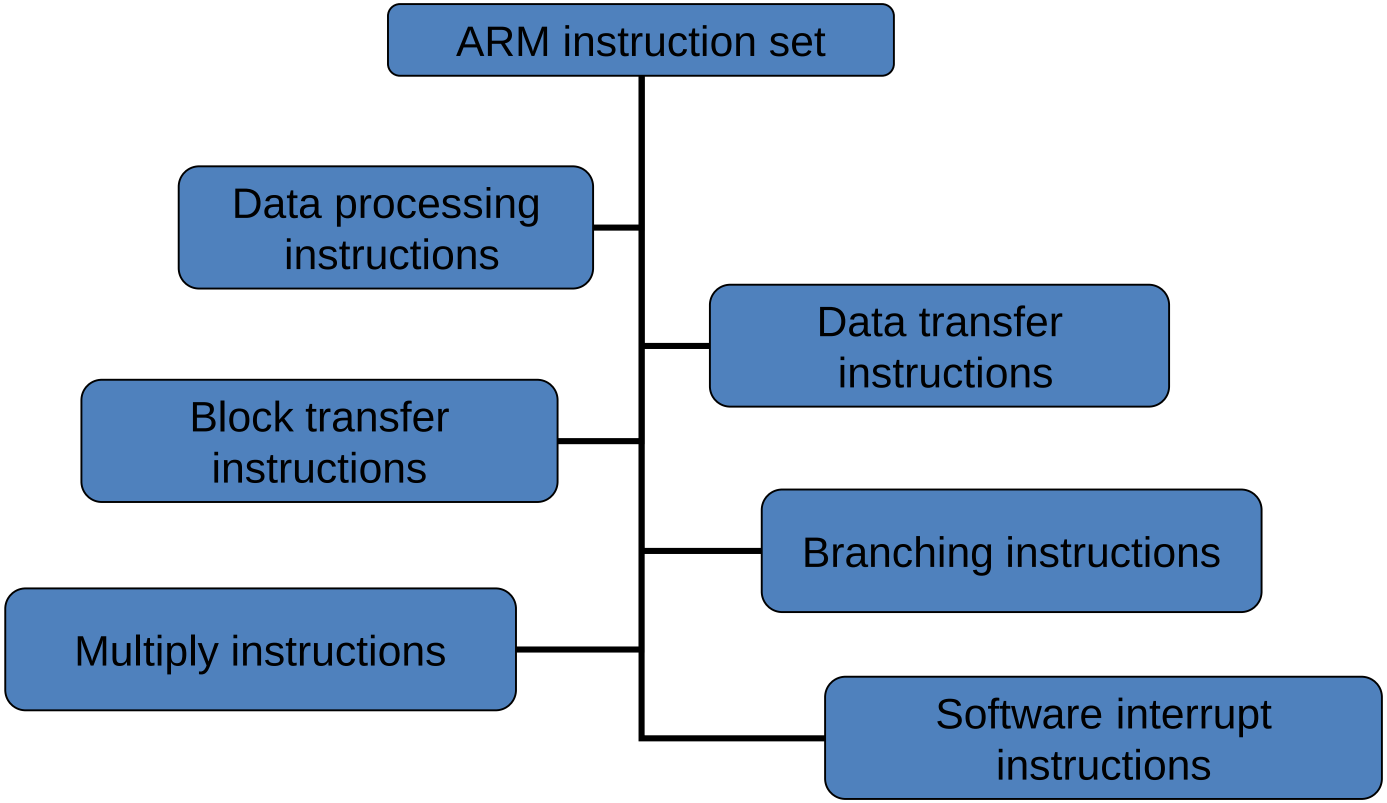
- 5 instructions
- 5 words
- 5 or 6 cycles

conditional

```
  CMP r0, #0
  ADDEQ r1, r1,
#1
  ADDNE r2, r2,
#1
  ...
```

- 3 instructions
- 3 words
- 3 cycles

ARM Instruction Set (3)



ARM instruction set

Data processing instructions

Data transfer instructions

Block transfer instructions

Branching instructions

Multiply instructions

Software interrupt instructions

Data Processing Instructions

- Arithmetic and logical operations
- 3-address format:
  - Two 32-bit operands
    (op1 is register, op2 is register or immediate)
  - 32-bit result placed in a register
- Barrel shifter for op2 allows full 32-bit shift within instruction cycle

Data Processing Instructions (2)

- Arithmetic operations:
  - ADD, ADDC, SUB, SUBC, RSB, RSC
- Bit-wise logical operations:
  - AND, EOR, ORR, BIC
- Register movement operations:
  - MOV, MVN
- Comparison operations:
  - TST, TEQ, CMP, CMN

Data Processing Instructions (3)

Conditional codes

+

Data processing instructions

+

Barrel shifter

=

Powerful tools for efficient coded programs

Data Processing Instructions (4)

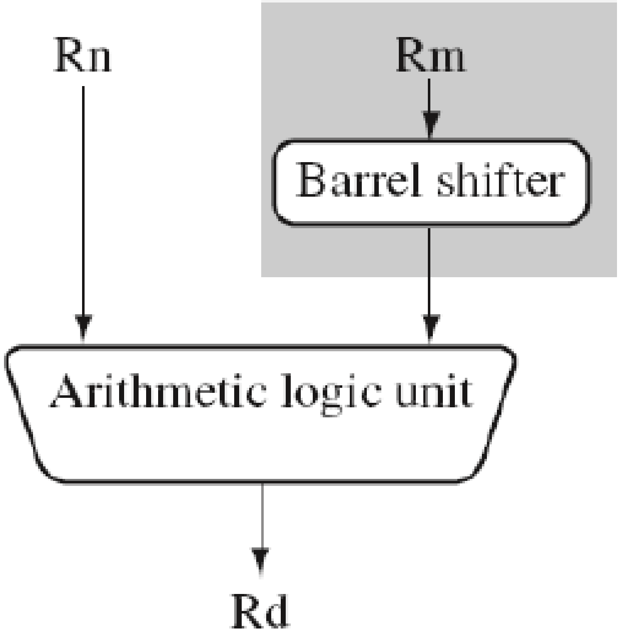
e.g.:

if (z==1) R1=R2+(R3*4)

compiles to

EQADDS R1,R2,R3, LSL #2

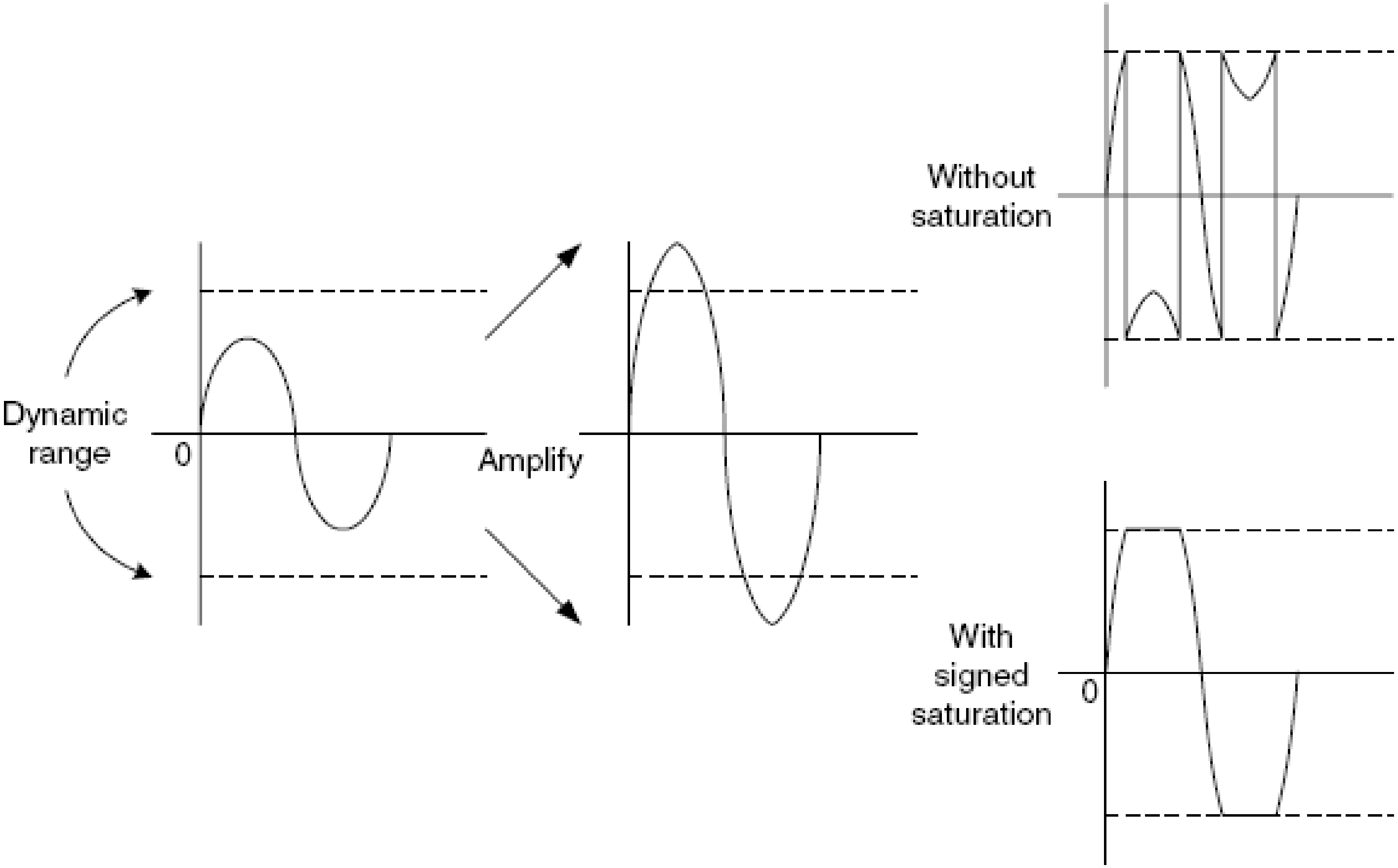
(SINGLE INSTRUCTION !)



Multiply Instructions

- Integer multiplication (32-bit result)
- Long integer multiplication (64-bit result)
- Built in Multiply Accumulate Unit (MAC)
- Multiply and accumulate instructions add product to running total

Saturated Arithmetic



Multiply Instructions

- Instructions:

MUL	Multiply	32-bit result
MULA	Multiply accumulate	32-bit result
UMULL	Unsigned multiply	64-bit result
UMLAL	Unsigned multiply accumulate	64-bit result
SMULL	Signed multiply	64-bit result
SMLAL	Signed multiply accumulate	64-bit result

Data Transfer Instructions

- Load/store instructions
- Used to move signed and unsigned
  Word, Half Word and Byte to and from registers
- Can be used to load PC
  (if target address is beyond branch instruction range)

LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	Store Half Word
LDRSH	Load Signed Half Word	STRSH	Store Signed Half Word
LDRB	Load Byte	STRB	Store Byte
LDRSB	Load Signed Byte	STRSB	Store Signed Byte

Addressing Modes

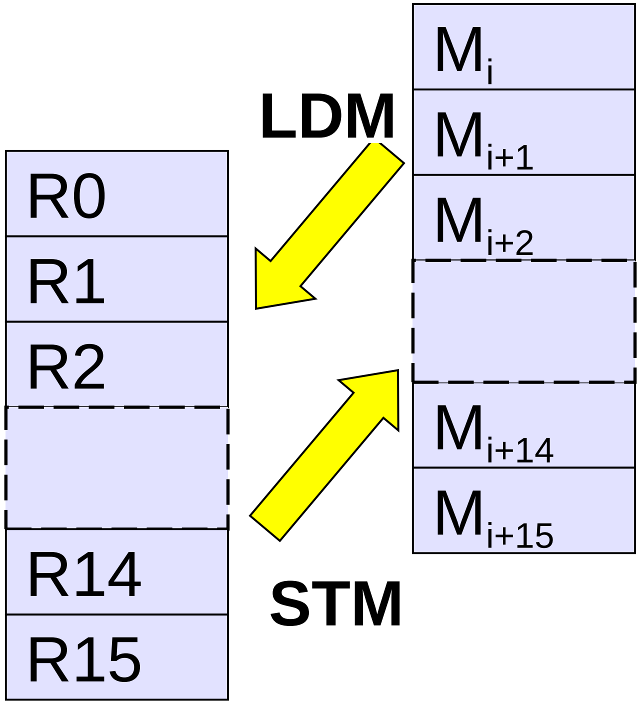
- Offset Addressing
  - Offset is added or subtracted from base register
  - Result used as effective address for memory access
  - [<Rn>, <offset>]
- Pre-indexed Addressing
  - Offset is applied to base register
  - Result used as effective address for memory access
  - Result written back into base register
  - [<Rn>, <offset>]!
- Post-indexed Addressing
  - The address from the base register is used as the EA
  - The offset is applied to the base and then written back
  - [<Rn>], <offset>

<offset> options

- An immediate constant
  - #10
- An index register
  - <Rm>
- A shifted index register
  - <Rm>, LSL #<shift>

Block Transfer Instructions

- Load/Store Multiple instructions (*LDM/STM*)
- Whole register bank or a subset copied to memory or restored with single instruction

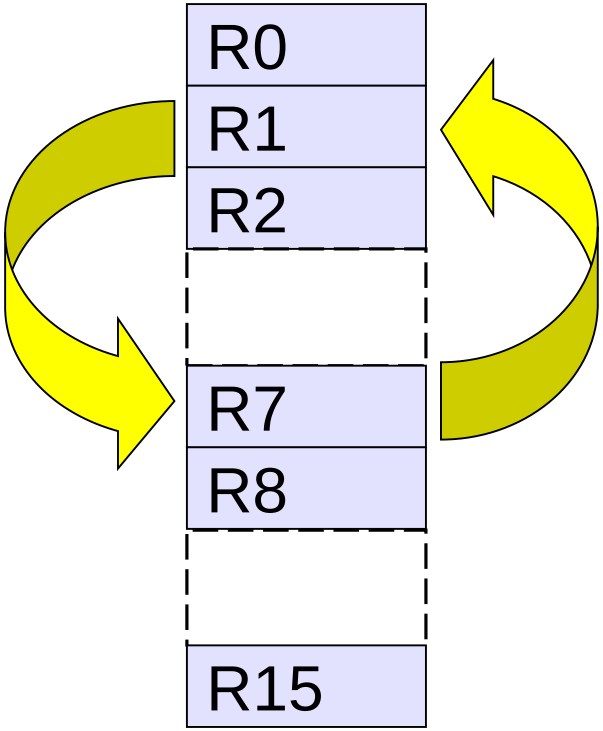


Swap Instruction

- Exchanges a word between registers
  - Two cycles

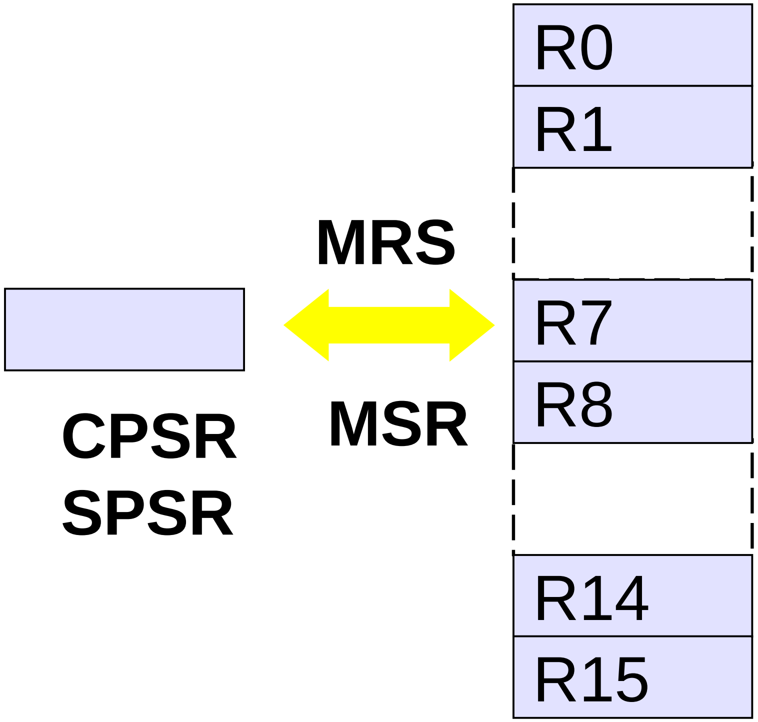
    but

  single atomic action
- Support for RT semaphores



Modifying the Status Registers

- Only indirectly
- *MSR* moves contents from CPSR/SPSR to selected GPR
- *MRS* moves contents from selected GPR to CPSR/SPSR
- Only in privileged modes



Software Interrupt

- *SWI* instruction
  - Forces CPU into supervisor mode
  - Usage: SWI #n

31 28	27 24	23 0
Cond	Opcode	Ordinal

- Maximum 2^{24} calls
- Suitable for running privileged code and making OS calls

Branching Instructions

- *Branch* (B):

  jumps forwards/backwards up to 32 MB

- *Branch link* (BL):

  same + saves (PC+4) in LR

- Suitable for function call/return

- Condition codes for conditional branches

IF-THEN Instruction

- Another alternative to execute conditional code is the new 16-bit IF-THEN (IT) instruction
  - no change in program flow
  - no branching overhead
- Can use with 32-bit Thumb-2 instructions that do not support the ‘S’ suffix
- Example:

```
CMP R1, R2          ; If R1 = R2
IT EQ               ; execute next (1st)
                    ; instruction
ADDEQ R2, R1, R0    ; 1st instruction
```

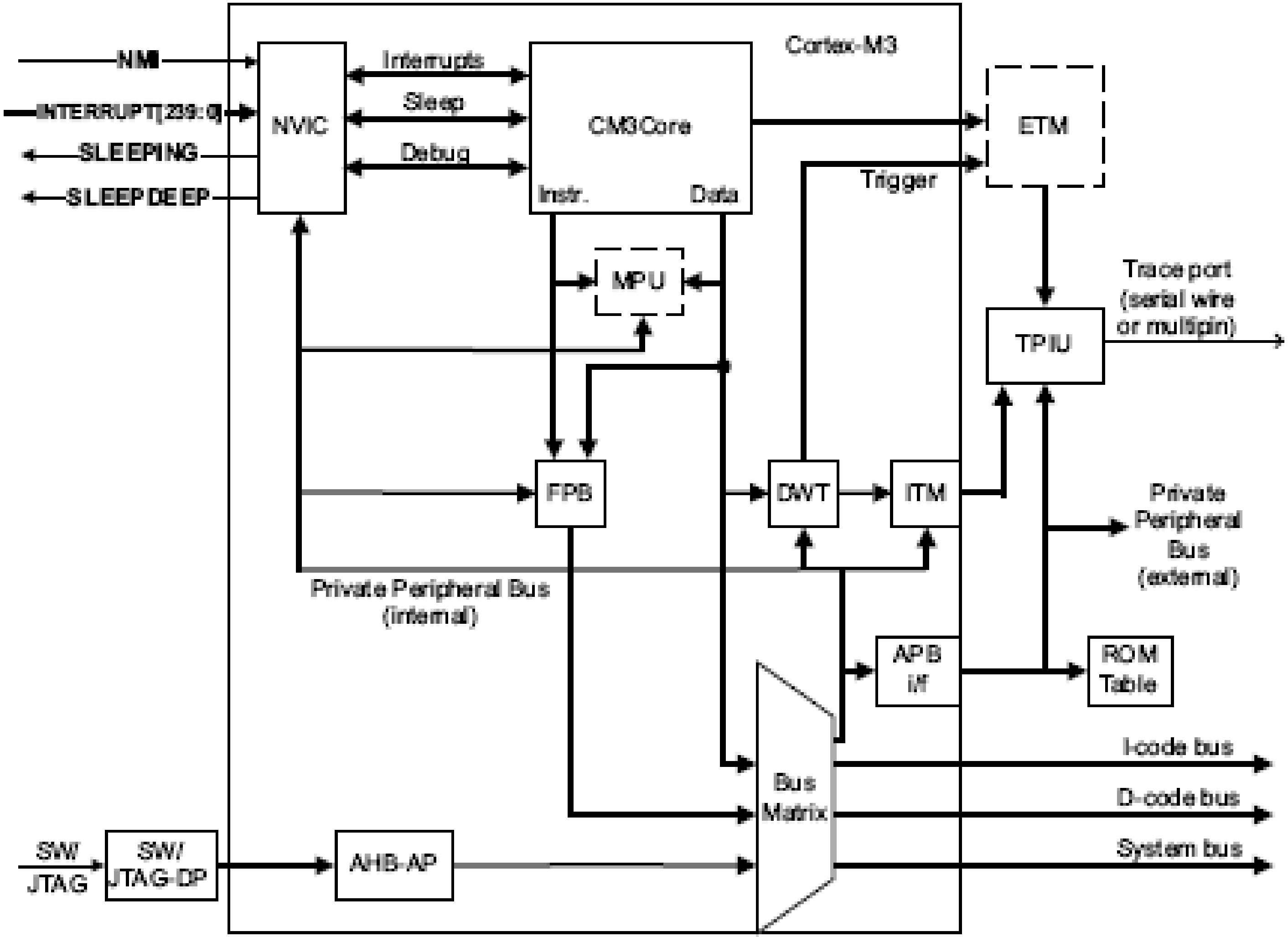
- The conditional codes can be extended up to 4 instructions

Barrier instructions

- Useful for multi-core & Self-modifying code

Instruction	Description
DMB	Data memory barrier; ensures that all memory accesses are completed before new memory access is committed
DSB	Data synchronization barrier; ensures that all memory accesses are completed before next instruction is executed
ISB	Instruction synchronization barrier; flushes the pipeline and ensures that all previous instructions are completed before executing new instructions

Backup


NMI
INTERRUPT[239:0]
SLEEPING
SLEEPDEEP
NVIC
Interrupts
Sleep
Debug
CM3Core
Instr.
Data
Cortex-M3
ETM
Trigger
MPU
Trace port
(serial wire
or multipin)
TPIU
FPB
DWT
ITM
Private
Peripheral
Bus
(external)
Private Peripheral Bus
(internal)
APB
i/f
ROM
Table
Bus
Matrix
I-code bus
D-code bus
System bus
SW/
JTAG
SW/
JTAG-DP
AHB-AP