# Introduction to ARM Processors

# OUTLINE

-Background

-ARM Microprocessor

- ARM Architecture,
- Assembly Language Programming
- Instruction Set

# BACKGROUND

- Architectural features of embedded processor
- **General rules (with exceptions):**
  1. Designed for efficiency (vs. ease of programming)
  2. Huge variety of processors (resulting from 1.)
  3. Harvard architecture
  4. Heterogeneous register sets
  5. Limited instruction-level parallelism or VLIW ISA
  6. Different operation modes (saturating arithmetic, fixed point)
  7. Specialised microcontroller & DSP instructions (bit-field addressing, multiply/accumulate, bit-reversal, modulo addressing)
  8. Multiple memory banks
- 9. No “fat” (MMU, caches, memory protection, target buffers, complex pipeline logic, ...)
- **These features have to be known to the compiler!**

# ARM Concept

- What is ARM?
  - Advanced RISC Machine
  - Acorn and VLSI Technology built in 1990/11
  - RISC
  - IP Core
  - T.I. ，PHILIPS，INTEL……
  - RISC Microcontroller
    - ARM7、ARM9、ARM9E-S、StrongARM ARM10…..

ARM的產品是 IP Core, 業務是銷售晶片系統的核心技術IP，全球有許多大型IT公司採用ARM的技術，如TI, Intel。

ARM的專利收入主要來自專利授權金以及按比例收取產品的專利使用費

# ARM Concept

- Why ARM?
  - Low power、Low cost、Tiny
  - 8/16/32 bit microprocessor
  - Thumb mode
  - Namely
    - T：Thumb Mode
    - D：Debug interface (JTAG)
    - M：Multiplier
    - I：ICE interface (Trace、Break point)

# Why ARM here?

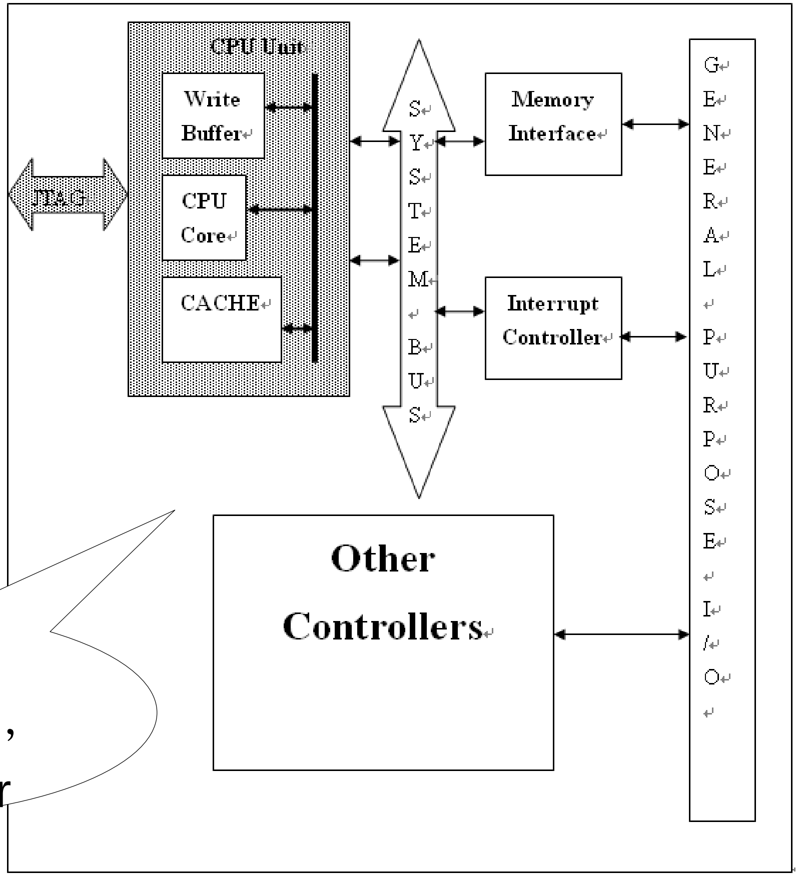
- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS / watt)
- Several interesting extensions available or in development like Thumb instruction set and Jazelle Java machine
  - http://www.arm.com/armtech/jazelle?OpenDocument

# ARM processor

- ARM is a family of RISC architectures.
- "ARM" is the abbreviation of "Advanced RISC Machines".
- ARM does not manufacture its own VLSI devices.
  - linceses
- ARM7- von Neuman Architecture
- ARM9 – Harvard Architecture

# ARM vs. SoC

- Architecture of ARM and SoC




ARM核心就是個CPU，SoC則是把系統要的功能全放到CPU內，可以提供特定用途的單晶片IC。以個人電腦爲例，將一部電腦除了電源外，皆轉變到一顆IC中。

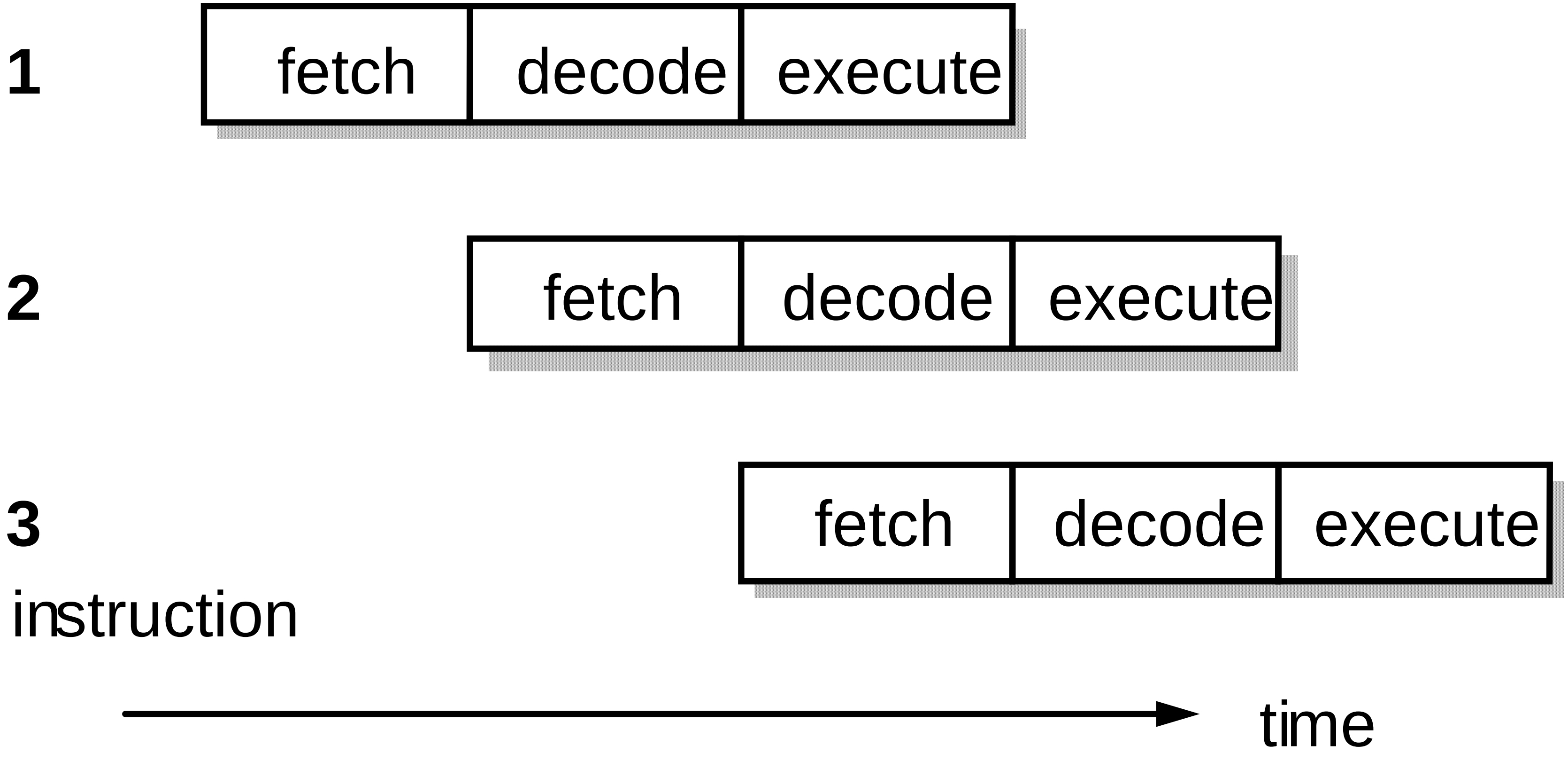
Ex:
LAN controller，
LCD controller

| | Cache Size (Inst/Data) | Tightly Coupled Memory | Memory Manage -ment | AHB Bus Interface | Thumb | DSP | Jazelle | Clock MHz ** |
|---|---|---|---|---|---|---|---|---|
| **Embedded Cores** | | | | | | | | |
| ARM7TDMI | No | No | No | Yes* | Yes | No | No | 133 |
| ARM7TDMI-S | No | No | No | Yes* | Yes | No | No | 100-133 |
| ARM7EJ-S | No | No | No | Yes* | Yes | Yes | Yes | 100-133 |
| ARM966E-S | No | Yes | No | Yes | Yes | Yes | No | 230-250 |
| ARM940T | 4K/4K | No | MPU | Yes* | Yes | No | No | 180 |
| ARM946E-S | Variable | Yes | MPU | Yes | Yes | Yes | No | 180-210 |
| ARM1026EJ-S | Variable | Yes | MMU+MPU | dual AHB | Yes | Yes | Yes | 266-325 |
| **Platform Cores** | | | | | | | | |
| ARM720T | 8K unified | No | MMU | Yes | Yes | No | No | 100 |
| ARM920T | 16K/16K | No | MMU | Yes* | Yes | No | No | 250 |
| ARM922T | 8K/8K | No | MMU | Yes* | Yes | No | No | 250 |
| ARM926EJ-S | Variable | Yes | MMU | dual AHB | Yes | Yes | Yes | 220-250 |
| ARM1020E | 32K/32K | No | MMU | dual AHB | Yes | Yes | No | 325 |
| ARM1022E | 16K/16K | No | MMU | dual AHB | Yes | Yes | No | 325 |
| ARM1026EJ-S | Variable | Yes | MMU+MPU | dual AHB | Yes | Yes | Yes | 266-325 |
| **Secure Applications** | | | | | | | | |
| SC100 | No | No | MPU | No | Yes | No | No | 80 |
| SC110 | No | No | MPU | No | Yes | No | No | 80 |
| SC200 | Optional | No | MPU | No | Yes | Yes | Yes | 110 |
| SC210 | Optional | No | MPU | No | Yes | Yes | Yes | 110 |
| **Intel ARM-based Processors** | | | | | | | | |
| StrongARM | 16K/8K | No | MMU | N/A | No | No | No | 206 |
| Intel XScale | 32K/32K | No | MMU | N/A | Yes | Yes | No | 400 |

# Intel Xscale

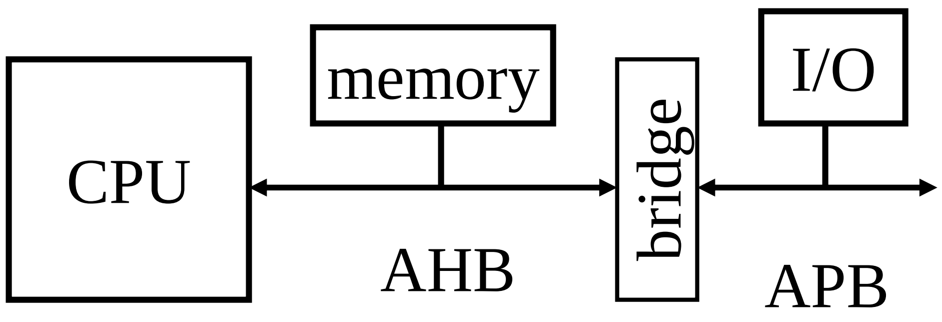
- ARM* Architecture Version 5TE ISA compliant.
  - ARM* Thumb Instruction Support
  - ARM* DSP Enhanced Instructions
- Low power consumption and high performance
- Intel® Media Processing Technology
  - Enhanced 16-bit Multiply
  - 40-bit Accumulator
- 32-KByte Instruction Cache
- 32-KByte Data Cache
- 2-KByte Mini Data Cache
- 2-KByte Mini Instruction Cache
- Instruction and Data Memory Management Units
- Branch Target Buffer
- Debug Capability via JTAG Port

# ARM single-cycle instruction 3-stage pipeline operation



# ARM busses

- AMBA:
  - Open standard.
  - Many external devices.
- Two varieties:
  - AMBA High-Performance Bus (AHB).
  - AMBA Peripherals Bus (APB).



# ARM instruction set

- ARM processor (operating) states
- ARM memory organization.
- ARM programming model.
- ARM assembly language.
- ARM data operations.
- ARM flow of control.
- C to assembly examples
- Exceptions
- Coprocessor instructions
- Summary

# Processor Operating States

- The ARM7TDMI processor has two operating states:
  – ARM - 32-bit, word-aligned ARM instructions are executed in this state.
  – Thumb -16-bit, halfword-aligned Thumb instructions are executed in this state.

- The operating state of the ARM7TDMI core can be switched between ARM state and Thumb state using the BX (branch and exchange) instructions

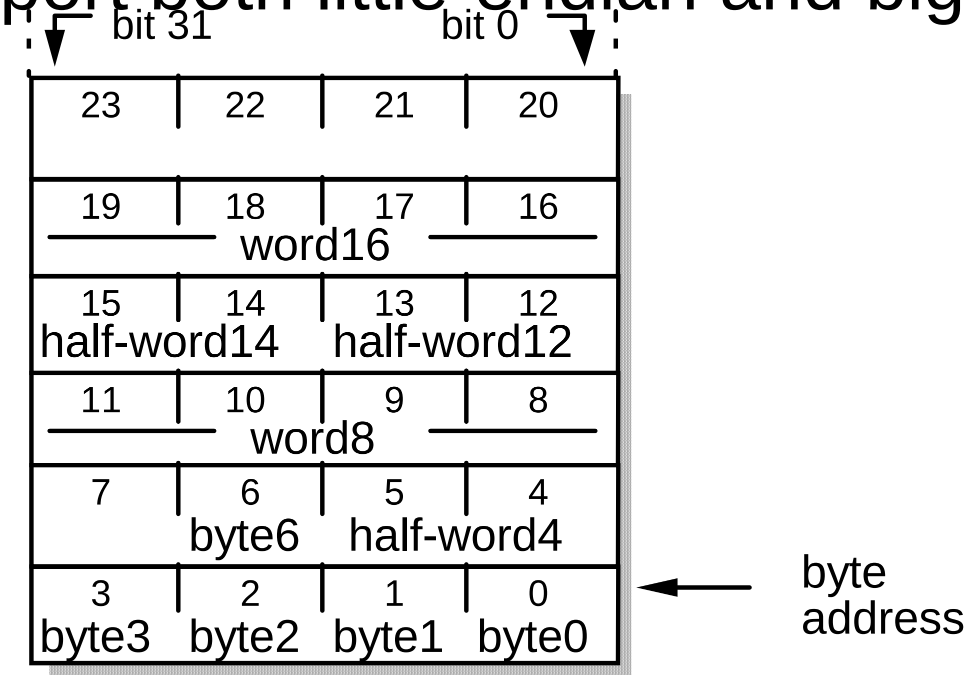
```
BX{<cond>}    <Rm>
```

where:

| | |
|---|---|
| `<cond>` | Is the condition under which the instruction is executed. The conditions are defined in *The condition field* on page A3-5. If `<cond>` is omitted, the `AL` (always) condition is used. |
| `<Rm>` | Holds the value of the branch target address. Bit[0] of Rm is 0 to select a target ARM instruction, or 1 to select a target Thumb instruction. |

# The Memory System

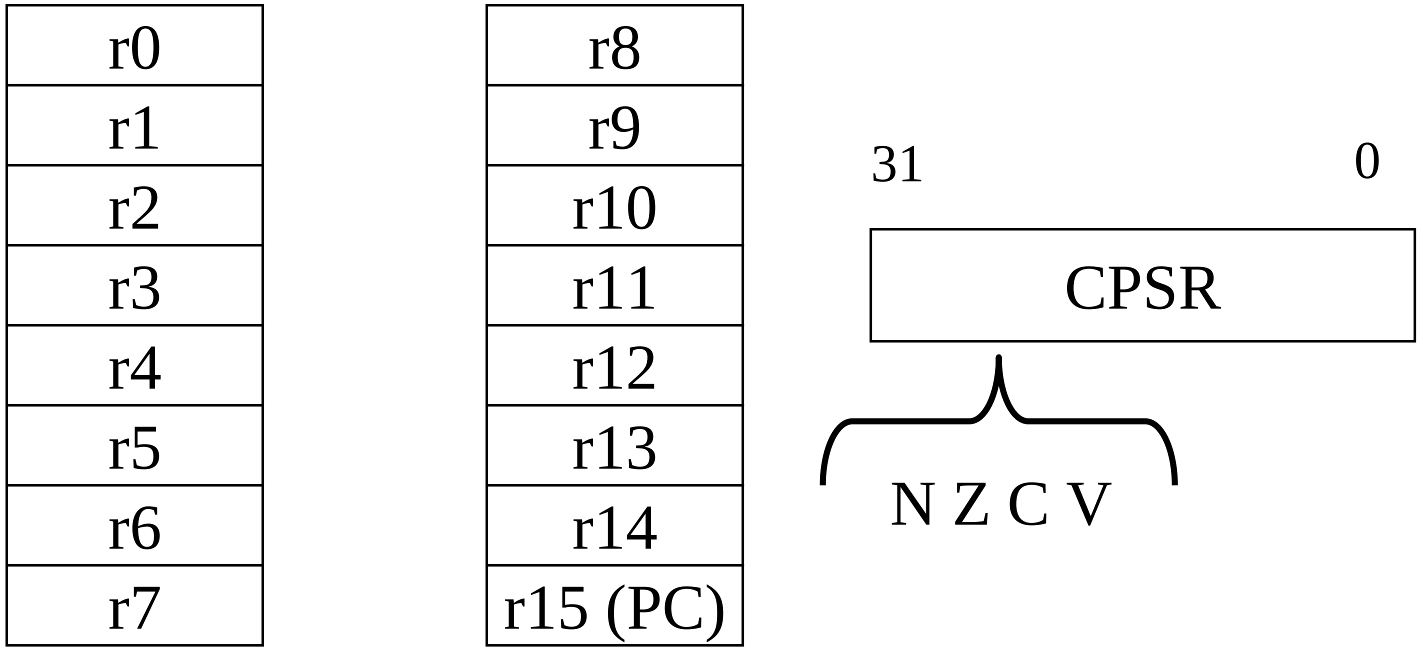
- 4 G address space
  - 8-bit bytes, 16-bit half-words, 32-bit words
  - Support both little-endian and big-endian



# Operating Modes

- The ARM7TDMI processor has seven modes of operations:
  - User mode(usr)
    - Normal program execution mode
  - Fast Interrupt mode(fiq)
    - Supports a high-speed data transfer or channel process.
  - Interrupt mode(irq)
    - Used for general-purpose interrupt handling.
  - Supervisor mode(svc)
    - Protected mode for the operating system.
  - Abort mode(abt)
    - implements virtual memory and/or memory protection
  - System mode(sys)
    - A privileged user mode for the operating system. (runs OS tasks)
  - Undefined mode(und)
    - supports a software emulation of hardware coprocessors
- Except user mode, all are known as privileged mode.

# ARM programming model

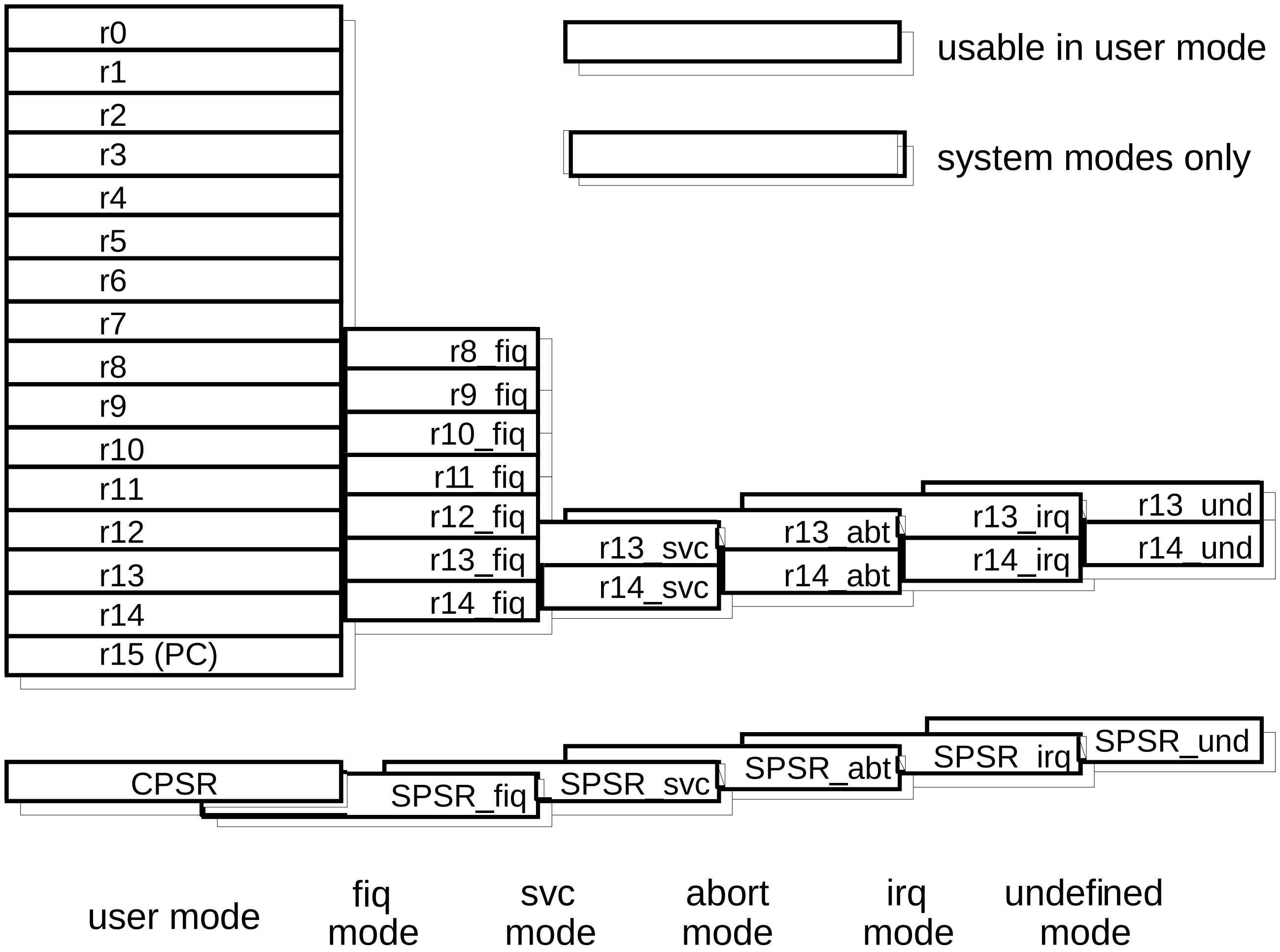



CPSR: Current Program Status Register
SPSR: Saved Program Status Register

# Registers

- 37 registers
  - 31 general 32 bit registers, including PC
  - 6 status registers
  - 15 general registers (R0 to R14), and one status registers and program counter are visible at any time – when you write user-level programs
    - R13 (SP)
    - R14 (LR)
    - R15 (PC)
- The visible registers depend on the processor mode
- The other registers (the banked registers) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing

# ARM Registers (1)



# Registers

- R0 to R15 are directly accessible
- R0 to R14 are general purpose
- R13: Stack point (sp) (in common)
  - Individual stack for each processor mode
- R14: Linked register (lr)
- R15 holds the Program Counter (PC)
- CPSR - Current Program Status Register contains condition code flags and the current mode bits
- 5 SPSRs (Saved Program Status Registers) which are loaded with CPSR when an exceptions occurs

# The Program Counter (R15)

- When the processor is executing in ARM state:
  - All instructions are 32 bits in length
  - All instructions must be word aligned
  - Therefore the PC value is stored in bits [31:2] with bits [1:0] equal to zero (as instruction cannot be halfword or byte aligned).
- R14 is used as the subroutine link register (LR) and stores the return address when Branch with Link (BL) operations are performed, calculated from the PC.
- Thus to return from a linked branch

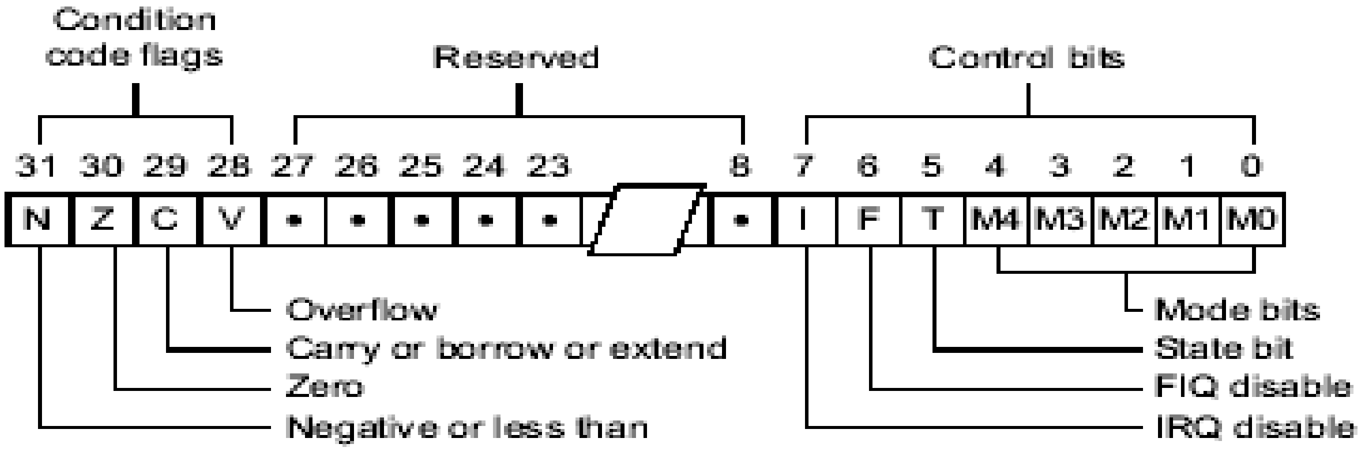
```
MOV r15,r14
MOV pc,lr
```

# Program Status Registers

- The ARM contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers.
- These register’s functions are:
  - Hold information about the most recently performed ALU operation.
  - Control the enabling and disabling of interrupts.
  - Set the processor operating mode

# Program Status Registers

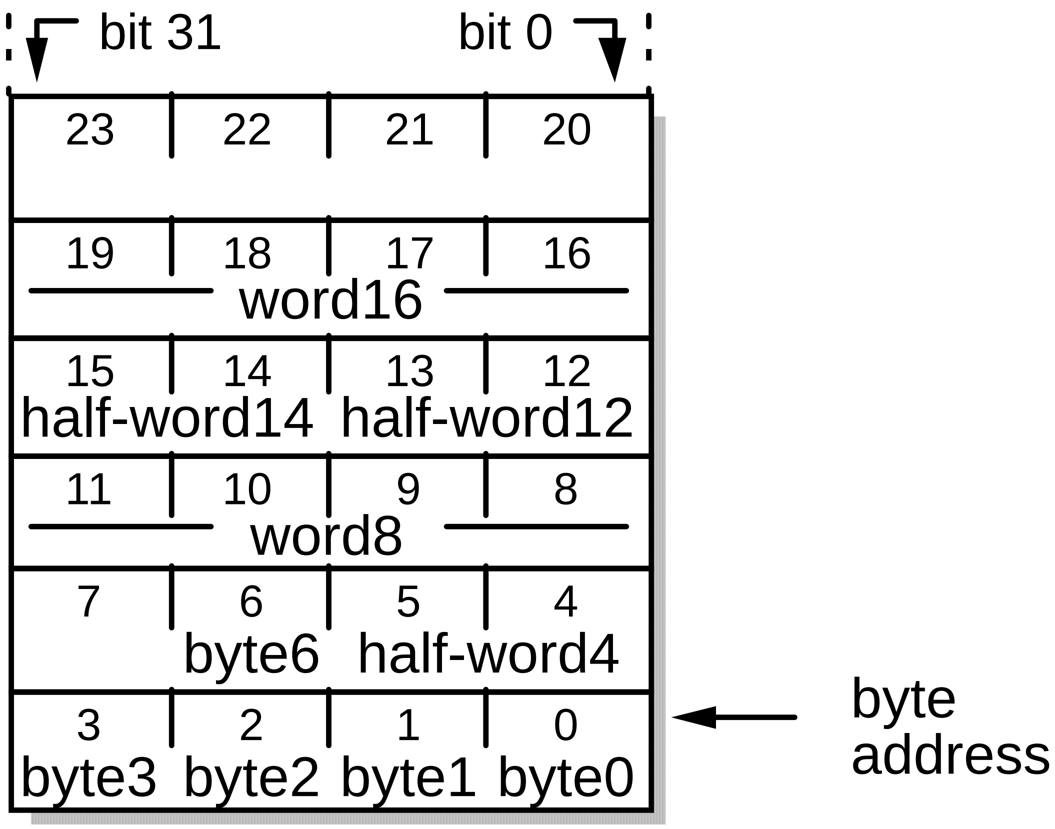
– The N, Z, C and V are condition code flags
  - may be changed as a result of arithmetic and logical operations in the processor
  - may be tested by all instructions to determine if the instruction is to be executed
  - N : Negative. Z : Zero. C : Carry. V : oVerflow
– The I and F bits are the interrupt disable bits
– The T bit is thumb bit
– The M0. M1. M2. M3 and M4 bits are the mode bits



# Program Counter (r15)

- When the processor is executing in ARM state:
  - All instructions are **32 bits wide**
  - All instructions must be **word aligned**
  - The **PC** value is stored in bits [31:2] with bits [1:0] undefined
  - Instructions cannot be halfword or byte aligned

# ARM Memory Organization



# Big Endian and Little Endian

**Big endian**

| Higher Address | 31 24 | 23 16 | 15 8 | 7 0 | Word Address |
|---|---|---|---|---|---|
| ↑ | 8 | 9 | 10 | 11 | 8 |
| | 4 | 5 | 6 | 7 | 4 |
| | 0 | 1 | 2 | 3 | 0 |
| Lower Address | | | | | |

- Most significant byte is at lowest address
- Word is addressed by byte address of most significant byte

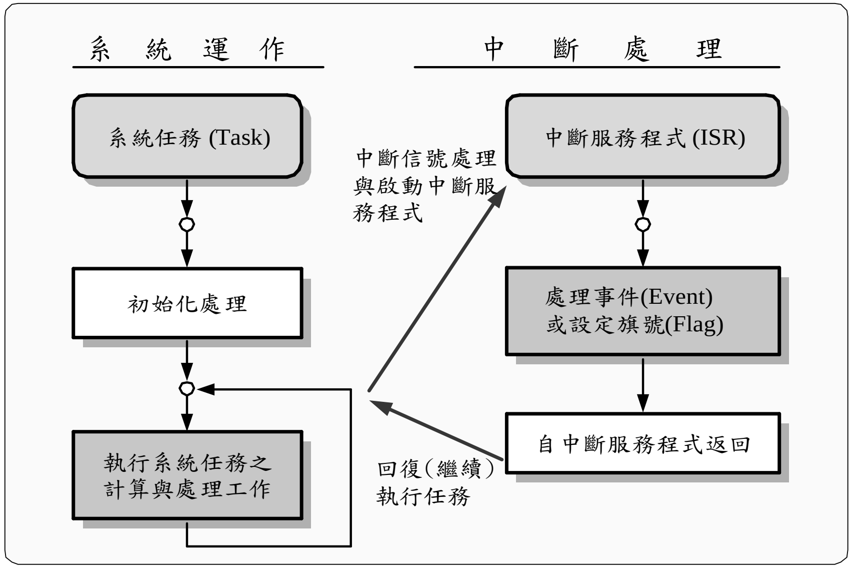
**Little endian**

| Higher Address | 31 24 | 23 16 | 15 8 | 7 0 | Word Address |
|---|---|---|---|---|---|
| ↑ | 11 | 10 | 9 | 8 | 8 |
| | 7 | 6 | 5 | 4 | 4 |
| | 3 | 2 | 1 | 0 | 0 |
| Lower Address | | | | | |

- Least significant byte is at lowest address
- Word is addressed by byte address of least significant byte

# Exceptions

- **Exceptions are usually used to handle unexpected events which arise during the execution of a program**




From 黃悅民等嵌入式系統設計-以ARM 處理器爲基礎之SoC平台

# Exception

- System Exception
  - CPU在執行時，愈到特殊的狀況而產生的例外，使用者完全無法對例外進行初始化、停止、或啓動
- Interrupt Exception
  - ARM CPU預留給系統建置者使用的中斷入口

# Exception Groups

- **Direct effect of executing an instruction**
  - SWI
  - Undefined instructions
  - Prefetch aborts (memory fault occurring during fetch)
- **A side-effect of an instruction**
  - Data abort (a memory fault during a load or store data access)
- **Exceptions generated externally**
  - Reset
  - IRQ
  - FIQ

# Exception Entry

- Change to the corresponding mode
- Save the address of the instruction following the exception instruction in **r14 of the new mode**
- Save the old value of CPSR in the **SPSR of the new mode**
- Disable IRQ
- If the exception is a FIQ, disables further FIQ
- Force PC to execute at the relevant vector address

# Exception Vector Addresses

| Exception | Mode | Vector address |
|---|---|---|
| Reset | SVC | 0x00000000 |
| Undefined instruction | UND | 0x00000004 |
| Software interrupt (SWI) | SVC | 0x00000008 |
| Prefetch abort (instruction fetch memory fault) | Abort | 0x0000000C |
| Data abort (data access memory fault) | Abort | 0x00000010 |
| IRQ (normal interrupt) | IRQ | 0x00000018 |
| FIQ (fast interrupt) | FIQ | 0x0000001C |

- Intel x86 – 0x00000 ~ 0x003FF (4 x 256)
- ARM – 0x000000 ~ 0x00001F

# Exception Return

- Any modified user registers must be restored
- Restore CPSR
- Resume PC in the correct instruction stream

# Exception Priorities

- Reset
- Data abort
- FIQ
- IRQ
- Prefetch abort
- SWI, undefined instruction

**Highest priority**

# Naming Rule of ARM

- **ARM {x} {y} {z} {T} {D} {M} {I} {E} {J} {F} {-S}**
  - x: series
  - y: memory management / protection unit
  - z: cache
  - T: Thumb decoder
  - D: JTAG debugger
  - M: fast multiplier
  - I: support hardware debug
  - E: enhance instructions (based on TDMI)
  - J: Jazelle
  - F: vector floating point unit
  - S: synthesiable, suitable for EDA tools

# Development of the ARM Architecture

1

2

3

Early ARM architectures

4

Halfword and signed halfword / byte support

System mode

SA-110

SA-1110

4T

Thumb instruction set

ARM7TDMI

ARM9TDMI

ARM720T

ARM940T

5TE

Improved ARM/Thumb Interworking

CLZ

Saturated maths

DSP multiply-accumulate instructions

ARM1020E

XScale

ARM9E-S

ARM966E-S

5TEJ

Jazelle

Java bytecode execution

ARM9EJ-S

ARM926EJ-S

ARM7EJ-S

ARM1026EJ-S

6

SIMD Instructions

Multi-processing

V6 Memory architecture (VMSA)

Unaligned data support

ARM1136EJ-S

reference: http://www.intel.com/education/highered/modelcurriculum.htm

# ARM assembly language

- Fairly standard assembly language:

```
        LDR r0,[r8] ; a comment
label   ADD r4,r0,r1
```

# ARM data types

- **32-bit word.**
- **Word can be divided into four 8-bit bytes.**
- **ARM addresses can be 32 bits long.**
- **Address refers to byte.**
  - Address 4 starts at byte 4.
- **Can be configured at power-up as either little- or bit-endian mode.**

# Instruction Set

- The ARM processor is very easy to program at the assembly level
- In this part, we will
  - **Look at ARM instruction set and assembly language programming at the user level**

# Notable Features of ARM Instruction Set

- The load-store architecture
- 3-address data processing instructions
- **Conditional execution of every instruction**
- The inclusion of every powerful load and store multiple register instructions
- Single-cycle execution of all instruction
- Open coprocessor instruction set extension

# Conditional Execution (1)

- One of the ARM's most interesting features is that each instruction is **conditionally executed**
- In order to indicate the ARM's conditional mode to the assembler, all you have to do is to append the appropriate condition to a mnemonic

```
    CMP     r0, #5
    BEQ     BYPASS
    ADD     r1, r1, r0
    SUB     r1, r1, r2
BYPASS
    …
```

⇒

```
    CMP       r0, #5
    ADDNE     r1, r1, r0
    SUBNE     r1, r1, r2
…
```

# Conditional Execution (2)

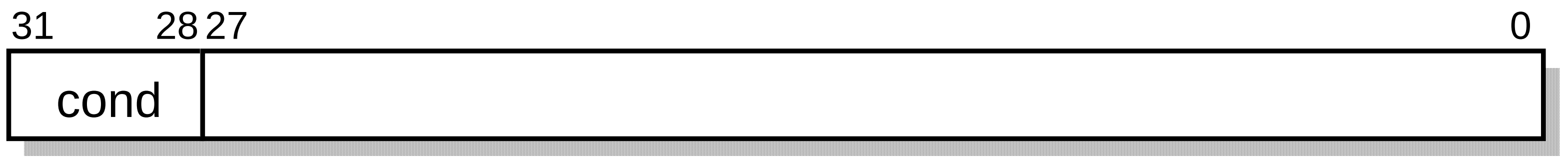
- The conditional execution code is faster and smaller

```
; if ((a==b) && (c==d))  e++;
;
; a is in register r0
; b is in register r1
; c is in register r2
; d is in register r3
; e is in register r4

    CMP    r0, r1
    CMPEQ  r2, r3
    ADDEQ  r4, r4, #1
```

# The ARM Condition Code Field

- Every instruction is conditionally executed
- Each of the 16 values of the condition field causes the instruction to be executed or skipped according to **the values of the N, Z, C and V flags in the CPSR**




**N: Negative Z: Zero C: Carry V: oVerflow**

# ARM Condition Codes

| Opcode [31:28] | Mnemonic extension | Interpretation | Status flag state for execution |
|---|---|---|---|
| 0000 | EQ | Equal / equals zero | Z set |
| 0001 | NE | Not equal | Z clear |
| 0010 | CS/HS | Carry set / unsigned higher or same | C set |
| 0011 | CC/LO | Carry clear / unsigned lower | C clear |
| 0100 | MI | Minus / negative | N set |
| 0101 | PL | Plus / positive or zero | N clear |
| 0110 | VS | Overflow | V set |
| 0111 | VC | No overflow | V clear |
| 1000 | HI | Unsigned higher | C set and Z clear |
| 1001 | LS | Unsigned lower or same | C clear or Z set |
| 1010 | GE | Signed greater than or equal | N equals V |
| 1011 | LT | Signed less than | N is not equal to V |
| 1100 | GT | Signed greater than | Z clear and N equals V |
| 1101 | LE | Signed less than or equal | Z set or N is not equal to V |
| 1110 | AL | Always | any |
| 1111 | NV | Never (do not use!) | none |

# Condition Field

- In ARM state, all instructions are conditionally executed according to the CPSR condition codes and the instruction's condition field
- Fifteen different conditions may be used
- **"Always" condition**
  - Default condition
  - May be omitted
- **"Never" condition**
  - The sixteen (1111) is reserved, and must not be used
  - May use this area for other purposes in the future