SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ITT204 - MICROCONTROLLER AND EMBEDDED SYSTEMS

II YEAR/ IV SEMESTER

UNIT III EMBEDDED SYSTEM CONCEPTS AND PROCESSORS

TOPIC – ARM Processors

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!**

OUTLINE

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?

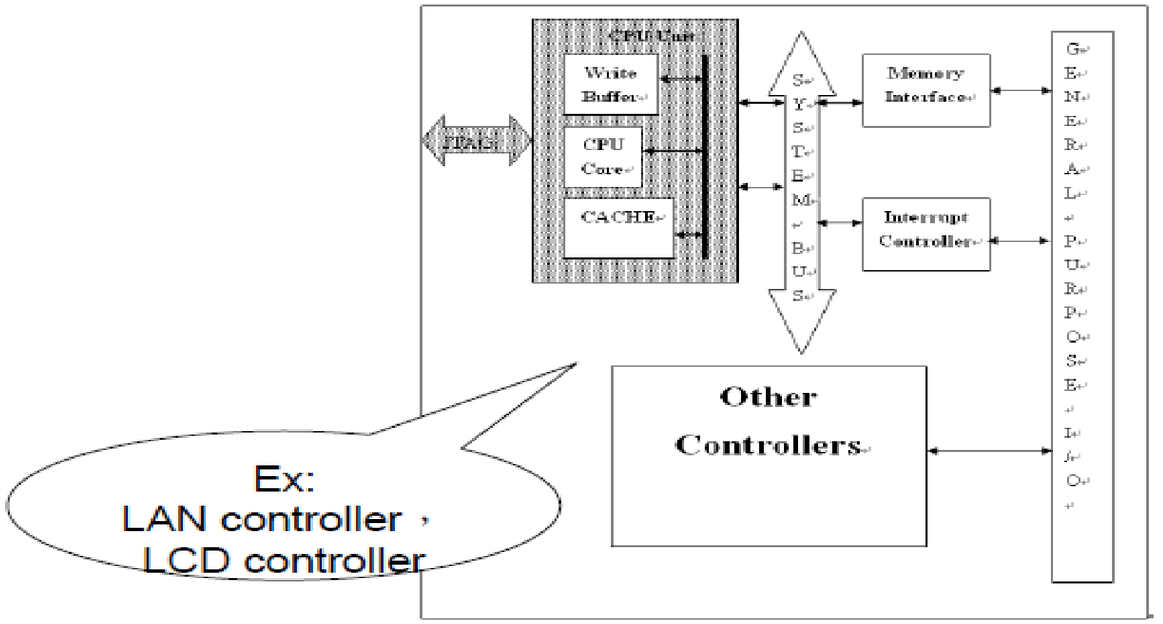
- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS / watt)
- Several interesting extensions available or in development like Thumb instruction set and Jazelle Java machine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

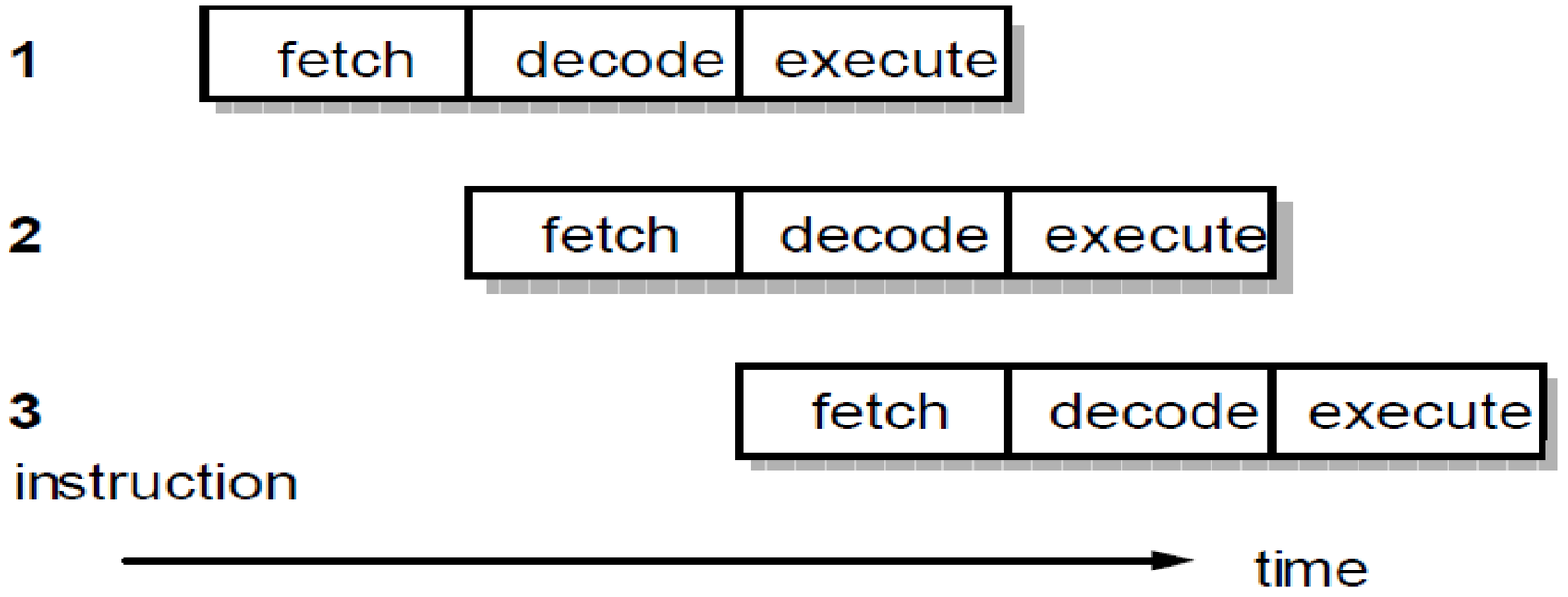
ARM vs. SoC

- Architecture of ARM and SoC



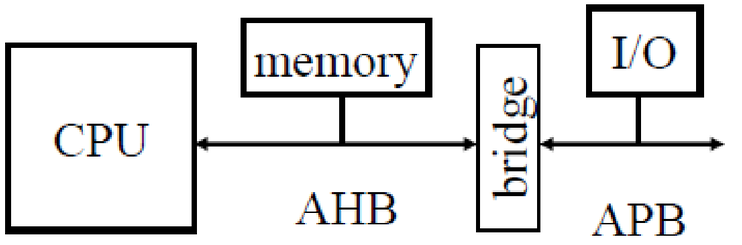
ARM核心就是個CPU，SoC則是把系統要的功能全放到CPU內，可以提供特定用途的單晶片IC。以個人電腦爲例，將一部電腦除了電源外，皆轉變到一顆IC中。

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).



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.

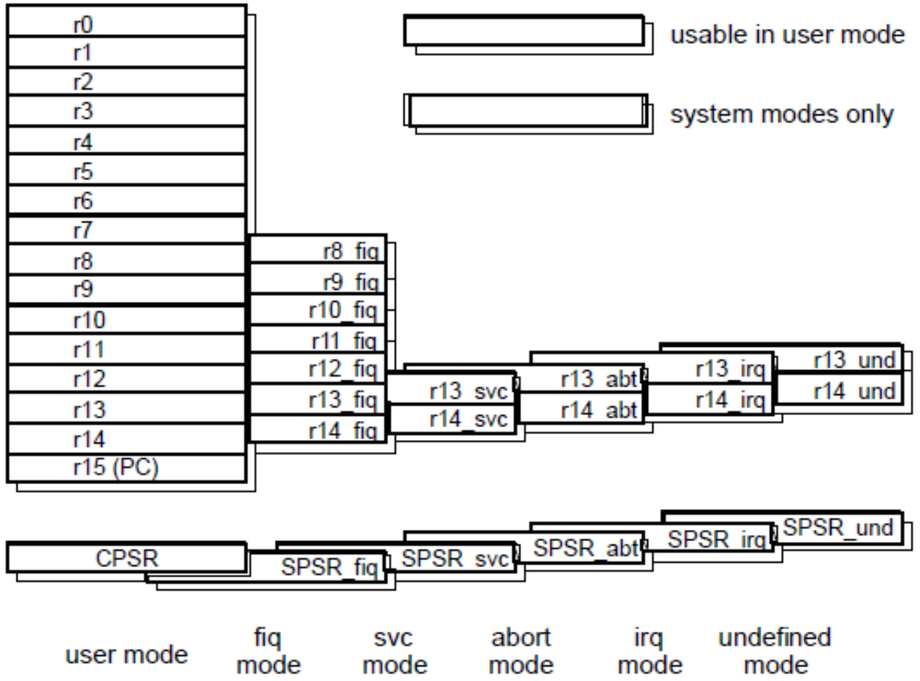
Operating Modes

- 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.

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

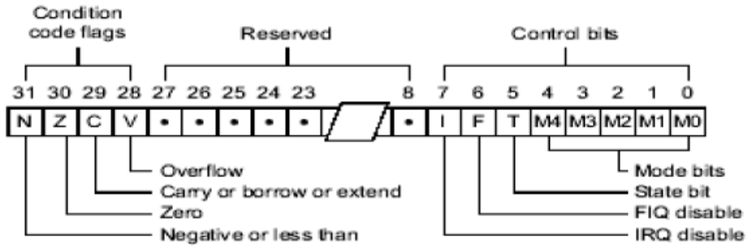
```
MOV r15,r14
MOV pc,lr
```

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



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

32

Exception Priorities

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

Highest priority

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.

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

ARM Instruction Set

- Data processing instructions
- Data transfer instructions
- Control flow instructions
- Writing simple assembly language programs

Data processing instructions

- Enable the programmer to perform **arithmetic** and **logical** operations on data values in registers
- The applied rules
  - All operands are 32 bits wide and come from registers or are specified as literals in the instruction itself
  - The result, if there is one, is 32 bits wide and is placed in a register

    (An exception: long multiply instructions produce a 64 bits result)
  - Each of the operand registers and the result register are independently specified in the instruction

    (This is, the ARM uses a '**3-address**' format for these instruction)

Arithmetic Operations

- These instructions perform binary arithmetic on two 32-bit operands
- The carry-in, when used, is the current value of the C bit in the CPSR

ADD r0, r1, r2	r0 := r1 + r2
ADC r0, r1, r2	r0 := r1 + r2 + C
SUB r0, r1, r2	r0 := r1 – r2
SBC r0, r1, r2	r0 := r1 – r2 + C – 1
RSB r0, r1, r2	r0 := r2 – r1
RSC r0, r1, r2	r0 := r2 – r1 + C – 1

Bit-Wise Logical Operations

- These instructions perform the specified boolean logic operation on each bit pair of the input operands

```
r0[i] := r1[i] OP_logic r2[i]     for i in [0..31]
```

AND r0, r1, r2	r0 := r1 AND r2
ORR r0, r1, r2	r0 := r1 OR r2
EOR r0, r1, r2	r0 := r1 XOR r2
BIC r0, r1, r2	r0 := r1 AND (NOT r2)

- **BIC stands for 'bit clear'**
- **Every '1' in the second operand clears the corresponding bit in the first operand**

Register Movement Operations

- These instructions ignore the first operand, which is omitted from the assembly language format, and simply move the second operand to the destination

MOV r0, r2	r0 := r2
MVN r0, r2	r0 := NOT r2

The '**MVN**' mnemonic stands for '**move negated**'

Comparison Operations

- These instructions do not produce a result, but just set the condition code bits (N, Z, C, and V) in the CPSR according to the selected operation

CMP r1, r2	**compare**	set cc on r1 – r2
CMN r1, r2	**compare negated**	set cc on r1 + r2
TST r1, r2	**bit test**	set cc on r1 AND r2
TEQ r1, r2	**test equal**	set cc on r1 XOR r2

Multiplies (1)

- A special form of the data processing instruction supports multiplication
- Some important differences
  - Immediate second operands are not supported
  - The result register must not be the same as the first source register
  - If the 'S' bit is set, the C flag is meaningless

```
MUL   r4, r3, r2    ; r4 := (r3 x r2)[31:0]
```

Data Transfer Instructions

- Move data between ARM registers and memory
- Three basic forms of data transfer instruction
  - Single register load and store instructions
  - Multiple register load and store instructions
  - Single register swap instructions

Single Register Load / Store Instructions (2)

LDR	Load a word into register	Rd ←mem32[address]
STR	Store a word in register into memory	Mem32[address] ←Rd
LDRB	Load a byte into register	Rd ←mem8[address]
STRB	Store a byte in register into memory	Mem8[address] ←Rd
LDRH	Load a half-word into register	Rd ←mem16[address]
STRH	Store a half-word in register into memory	Mem16[address] ←Rd
LDRSB	Load a signed byte into register	Rd ←signExtend(mem8[address])
LDRSH	Load a signed half-word into register	Rd ←signExtend(mem16[address])

Multiple Register Load / Store Instructions (2)

LDM	Load multiple registers
STM	Store multiple registers

Addressing mode	Description	Starting address	End address	Rn!
IA	Increment After	Rn	Rn+4*N-4	Rn+4*N
IB	Increment Before	Rn+4	Rn+4*N	Rn+4*N
DA	Decrement After	Rn-4*Rn+4	Rn	Rn-4*N
DB	Decrement Before	Rn-4*N	Rn-4	Rn-4*N

Addressing mode for multiple register load and store instructions

Single Register Swap Instructions (1)

- Allow a value in a register to be exchanged with a value in memory
- Effectively do both a load and a store operation in one instruction
- They are little used in user-level programs
- Atomic operation
- Application
  - Implement semaphores (multi-threaded / multi-processor environment)

87

Control Flow Instructions

- Determine which instructions get executed next

```
        B       LABEL
        …
        …
LABEL   …
```

```
        MOV     r0, #0      ; initialize counter
LOOP    …
        ADD     r0, r0, #1  ; increment loop counter
        CMP     r0, #10     ; compare with limit
        BNE     LOOP        ; repeat if not equal
        …                   ; else fall through
```

Branch Conditions

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
~~BAL~~	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

THANK YOU