Thiết kế hệ thống nhúng-EE4251

GV: ĐÀO ĐỨC THỊNH

KHOA TỰ ĐỘNG HOÁ-TRƯỜNG ĐIỆN ĐIỆN TỬ-ĐHBK HN

ARM

- Mô tả chung ARM Ltd
- Cấu trúc ARM
- Tập lệnh
- Thiết kế hệ thống dựa trên ARM
- Công cụ phát triển

Mô tả chung ARM Ltd

- Acorn Computers Limited, based in Cambridge, England.
- In 1979, Acorn Atom released. Used the Rockwell 6502 1Mhz 8 bit CPU. Used in Apple II.
- Acorn makes agreement with the BBC (British Broadcasting Corporation),for a new computer design
- In 1981, BBC “The Computer Programme”project need to have a computer to demonstrate various tasks including “teletext/telesoftware, comms, controlling hardware, programming, artificial intelligence, graphics, sound and music, etc.
- As Acorn can’t find any processor ready on the market is acceptable for their needs, they wanted to design a new processor.
- Influenced by the Berkeley RISC I CPU.
- After some custom modifications by Acorn, a new RISC processor was designed
- The ARM (Advanced RISC Machine).

Mô tả chung ARM Ltd

Acorn - a Computer Manufacturer

1983:

- Acorn Limited:
- Dominant position in UK personal computer market with Rockwell/MOS Technology 6502 (8- Bit) CPU.

1983:

- 16- Bit CISC CPU´s slower than standard memory ports with long interrupt latencies

1983- 85:

- Acorn designed the first commercial RISC CPU:
- Acorn Risc Machine (ARM)

1990:

- Advanced Risc Machine was formed to broaden the market beyond Acorn´s product range

Mô tả chung ARM Ltd

1990:

- Startup with 12 engineers and 1 CEO
- No patents, no customers, very little money

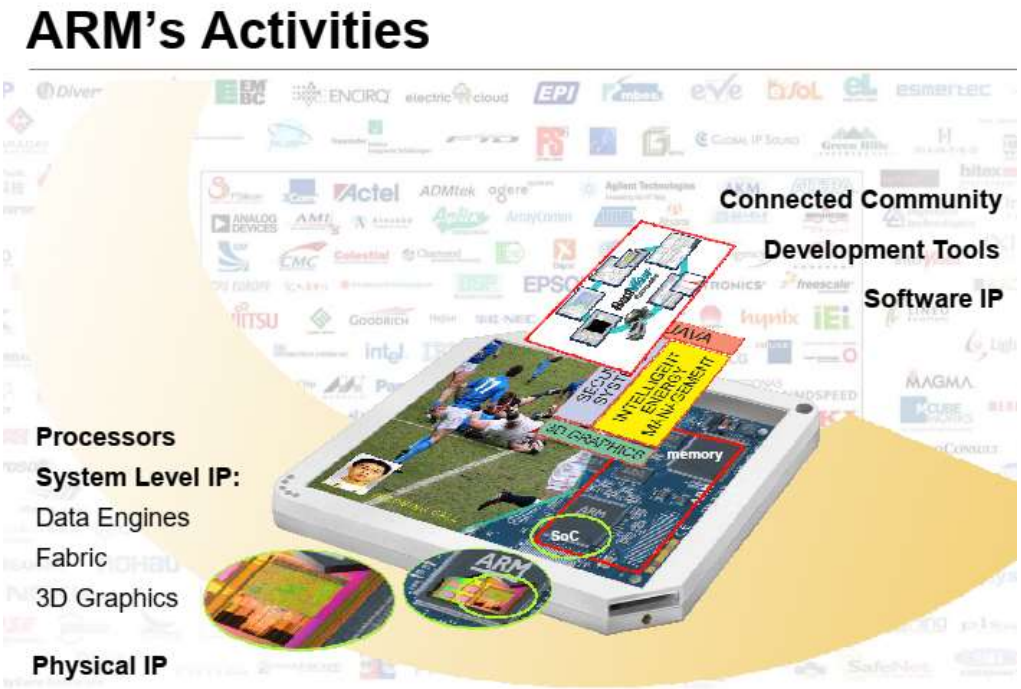
Mid- 1990s:

- T. I. licensed ARM7
- Incorporated into a chip for mobile phones

IPO Spring 1998

- 13 millionaires

Mô tả chung ARM Ltd

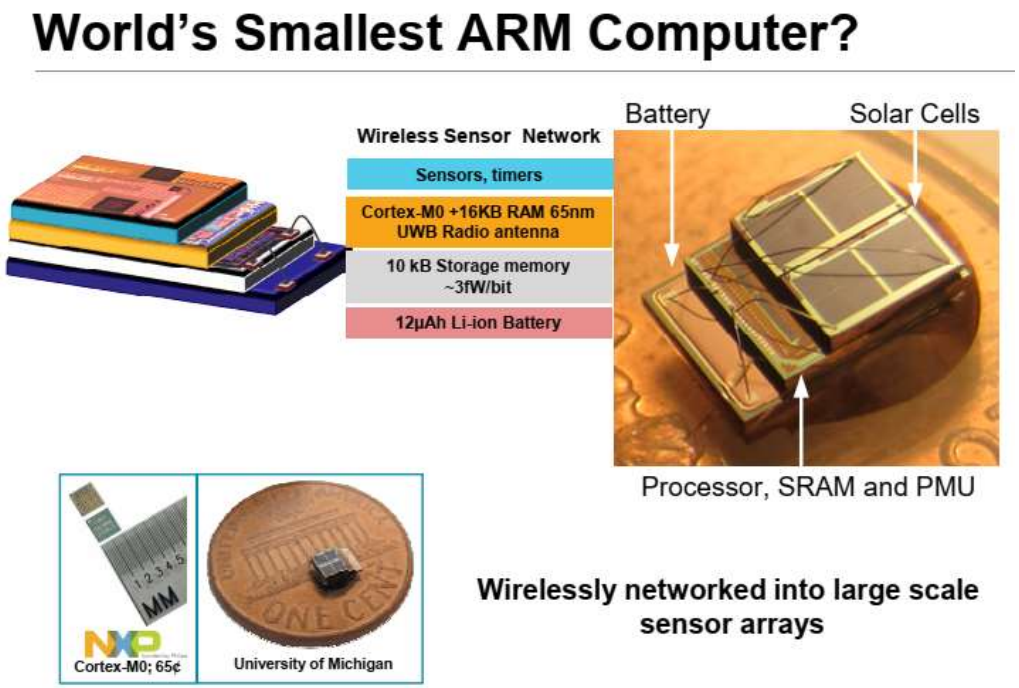


Mô tả chung ARM Ltd

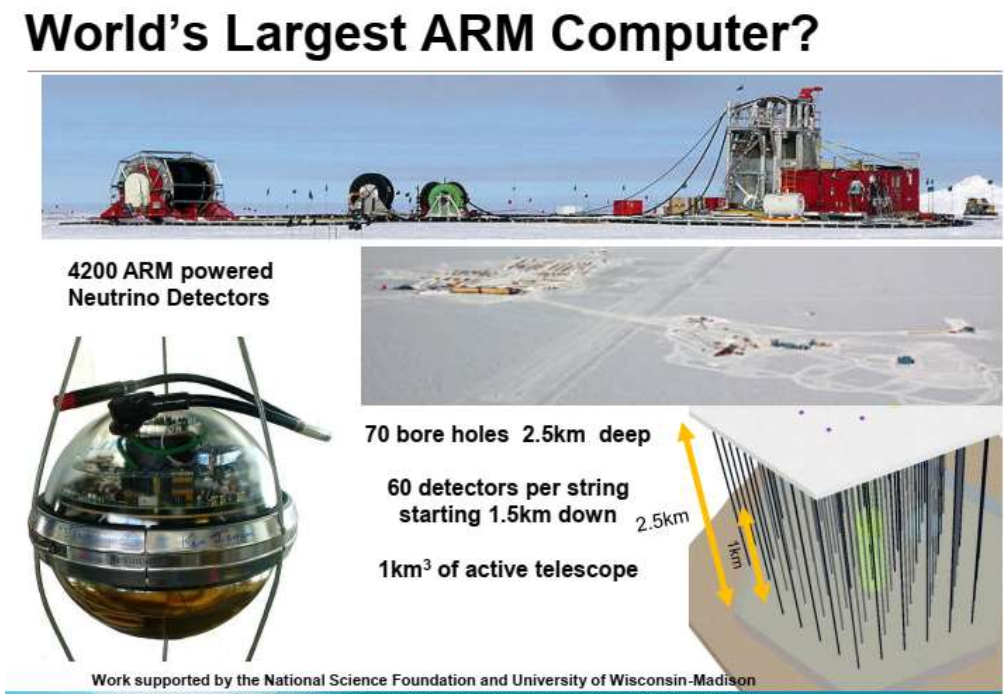
ARM Connected Community – 700+

Mô tả chung ARM Ltd

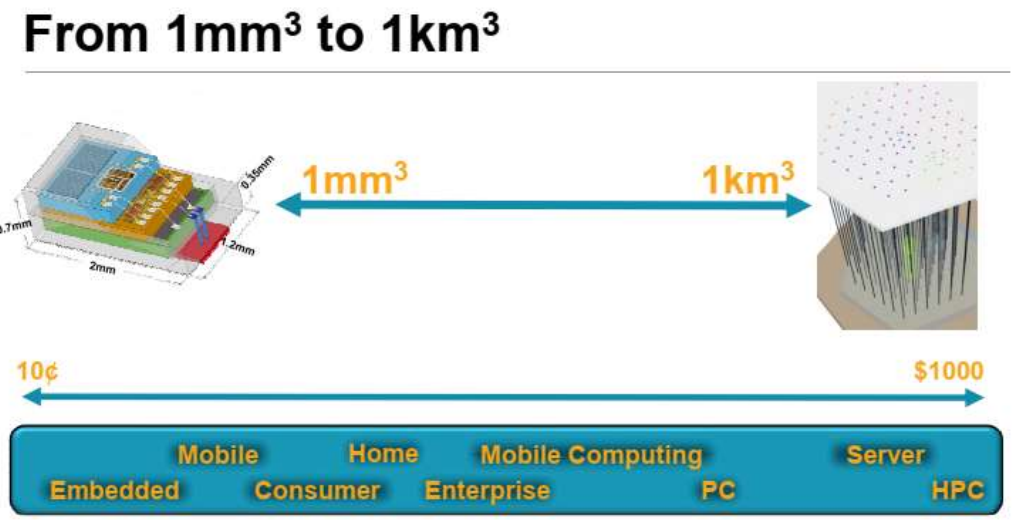
Mô tả chung ARM Ltd



Mô tả chung ARM Ltd



Mô tả chung ARM Ltd



Tại sao ARM?

- Một trong những lõi xử lý được cấp phép nhiều nhất và do đó phổ biến rộng rãi trên thế giới
  - Được sử dụng trong PDA, điện thoại di động, máy nghe nhạc đa phương tiện, máy chơi game cầm tay, TV kỹ thuật số và máy ảnh, TV kỹ thuật số và máy ảnh
  - ARM7: GBA, iPod
  - ARM9: NDS PSP Sony Ericsson BenQ , PSP, Sony Ericsson, BenQ
  - ARM11: Apple iPhone, Nokia N93, N800
  - 90% bộ xử lý RISC nhúng 32-bit tính đến năm 2009
- Được sử dụng đặc biệt trong các thiết bị di động do tính chất điện năng tiêu thụ thấp và hiệu suất hợp lý.

VXL ARM

- Một thiết kế đơn giản nhưng mạnh mẽ
- Một loạt thiết kế chia sẻ chung nguyên tắc thiết kế và tập lệnh

Đặt tên ARM

ARMxyzTDMIEJFS
– x: series
– y: MMU
– z: cache
– T: Thumb
– D: debugger
– M: Multiplier
– I: EmbeddedICE (built-in debugger hardware)
– E: Enhanced instruction
– J : Jazelle (JVM)
– F: Floating-point
– S : Synthesizible version (source code version for EDA tools)

Đặt tên ARM

- ARM7TDMI
  - 3 pipeline stages (fetch/decode/execute)
  - High code density/low power consumption
  - One of the most used ARM-version (for low-end systems)
  - All ARM cores after ARM7TDMI include TDMI even if they do not include TDMI in their labels
- ARM9TDMI
  - Compatible with ARM7
  - 5 stages (fetch/decode/execute/memory/write)
  - Separate instruction and data cache
- ARM11

Đặt tên ARM

ARM family attribute comparison.

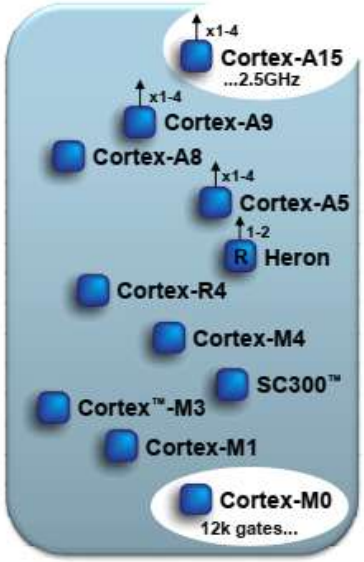
year	1995	1997	1999	2003
	ARM7	ARM9	ARM10	ARM11
Pipeline depth	three-stage	five-stage	six-stage	eight-stage
Typical MHz	80	150	260	335
mW/MHz[a]	0.06 mW/MHz	0.19 mW/MHz (+ cache)	0.5 mW/MHz (+ cache)	0.4 mW/MHz (+ cache)
MIPS[b]/MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 × 32	8 × 32	16 × 32	16 × 32

[a] Watts/MHz on the same 0.13 micron process.

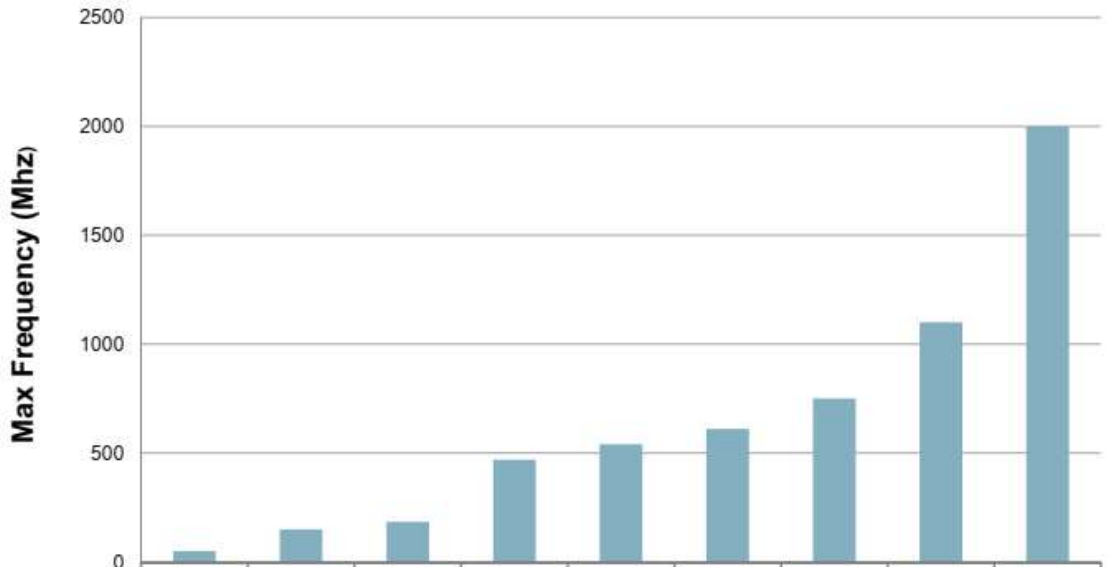
[b] MIPS are Dhrystone VAX MIPS.

Cấu trúc ARM

- ARM Cortex-**A** family (v7-A):
  - Applications processors for full OS and 3rd party applications
- ARM Cortex-**R** family (v7-R):
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-**M** family (v7-M):
  - Microcontroller-oriented processors for MCU and SoC applications



Cấu trúc ARM



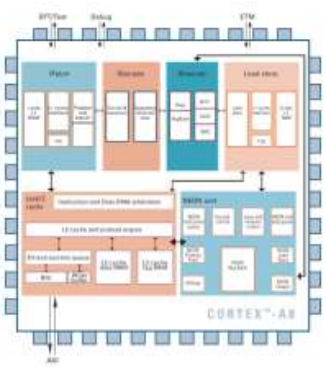
	Cortex-M0	Cortex-M3	ARM7	ARM926	ARM1026	ARM1136	ARM1176	Cortex-A8	Cortex-A9 Dual-core
Max Freq (MHz)	50	150	184	470	540	610	750	1100	2000
Min Power (mW/MHz)	0.012	0.06	0.35	0.235	0.36	0.335	0.568	0.43	0.5

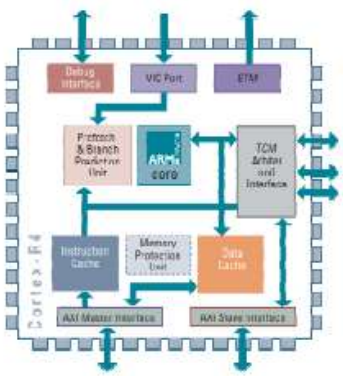
Cấu trúc ARM

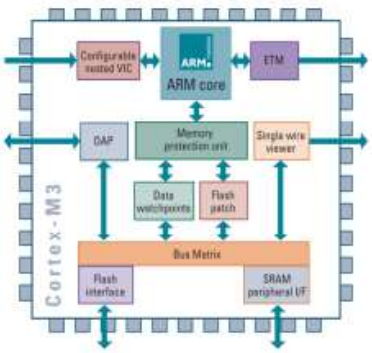
Cortex-A8
- Architecture v7A
- MMU
- AXI
- VFP & NEON support

Cortex-R4
- Architecture v7R
- MPU (optional)
- AXI
- Dual Issue

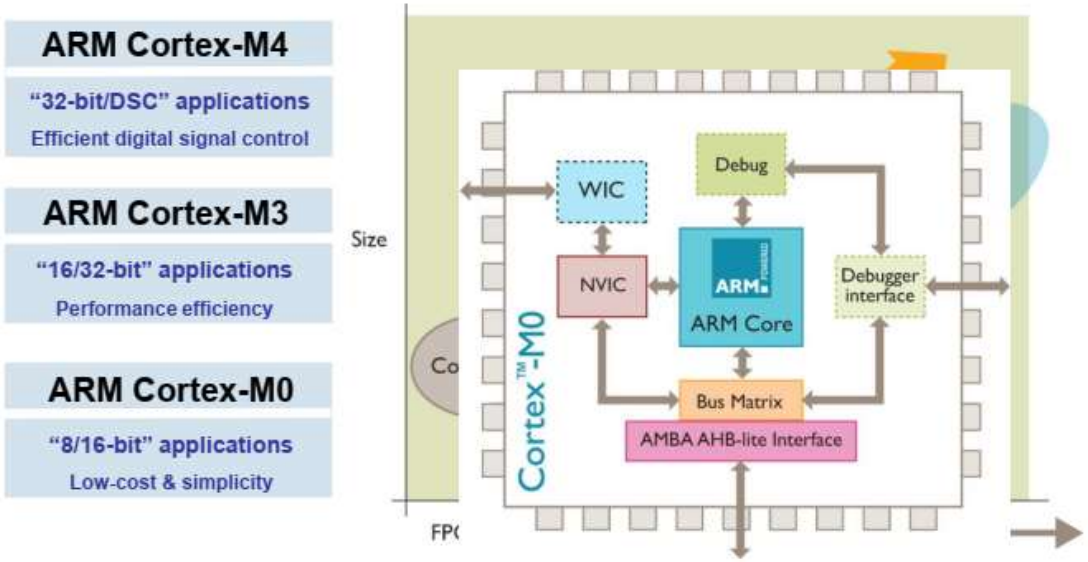
Cortex-M3
- Architecture v7M
- MPU (optional)
- AHB Lite & APB







Cấu trúc ARM


ARM Cortex-M4
"32-bit/DSC" applications
Efficient digital signal control
ARM Cortex-M3
"16/32-bit" applications
Performance efficiency
ARM Cortex-M0
"8/16-bit" applications
Low-cost & simplicity
Size
WIC
Debug
NVIC
ARM Core
Debugger interface
Bus Matrix
AMBA AHB-lite Interface
Cortex™-M0

Cấu trúc ARM

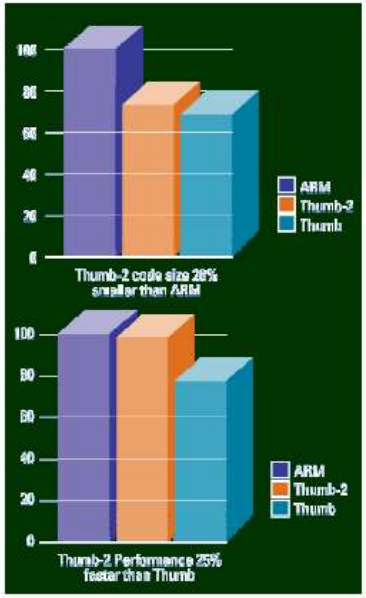
ARM Cortex-M0 processor features	Full product options	"M0_DS" implementation
Zero jitter 32-bit RISC core	✔	✔
AMBA AHB-lite interface	✔	✔
ARMv6-M instruction set architecture	✔	✔
NVIC Interrupt controller	✔	✔
Interrupt line configurations	1 to 32	16 only
Debug (SWD, JTAG) option	✔	
Up to 4 breakpoints, 2 watchpoints	✔	
Low power optimisations (ACG)	✔	
Multiple power domain support with WIC	✔	
Fast multiplier (1 cycle) option	✔	
System timer	✔	✔
Area (gates)	**12k – 25k**	**16K**

Minimum usable

Cấu trúc ARM

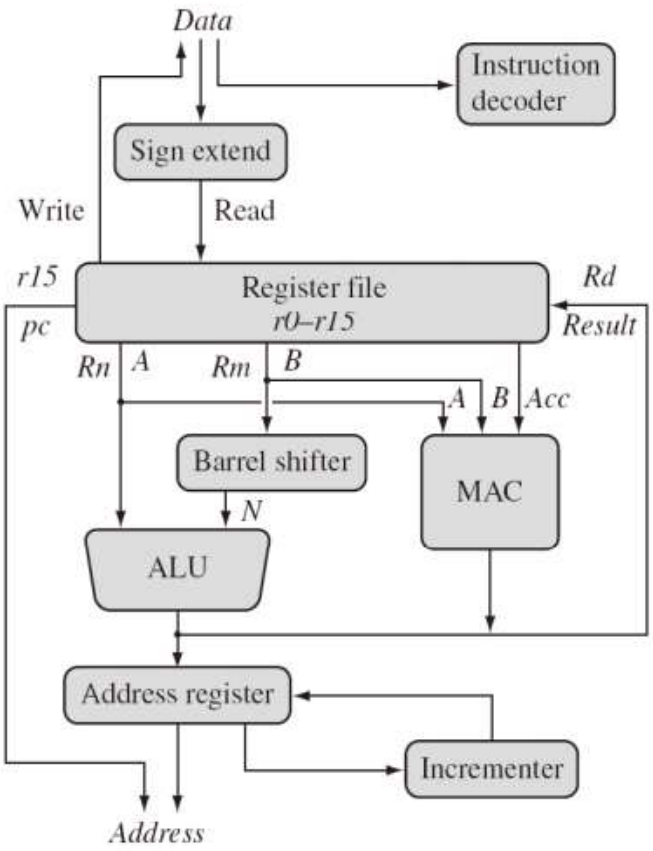
The Thumb-2 instruction set

- Variable-length instructions
  - ARM instructions are a fixed length of 32 bits
  - Thumb instructions are a fixed length of 16 bits
  - Thumb-2 instructions can be either 16-bit or 32-bit
- Thumb-2 gives approximately 26% improvement in code density over ARM
- Thumb-2 gives approximately 25% improvement in performance over Thumb



Cấu trúc ARM

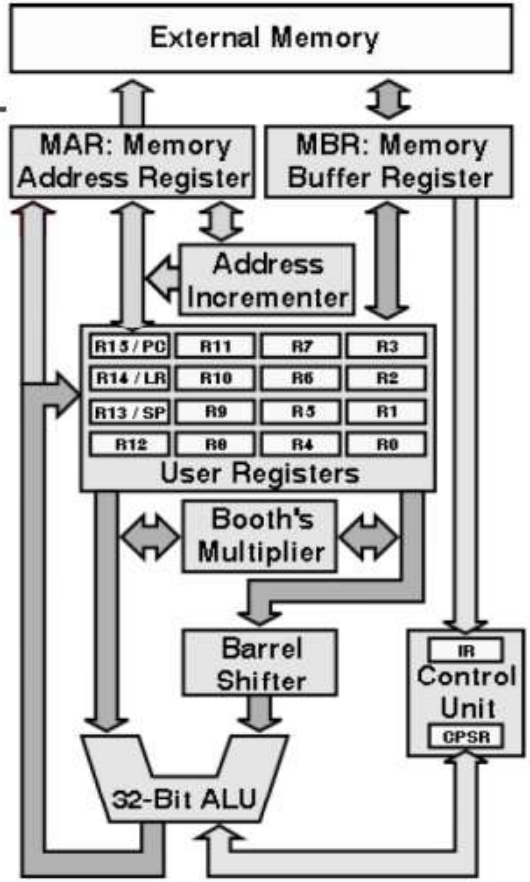
ARM architecture



Cấu trúc ARM

ARM architecture

- Load/store architecture
- A large array of uniform registers
- Fixed-length 32-bit instructions
- 3-address instructions



Cấu trúc ARM

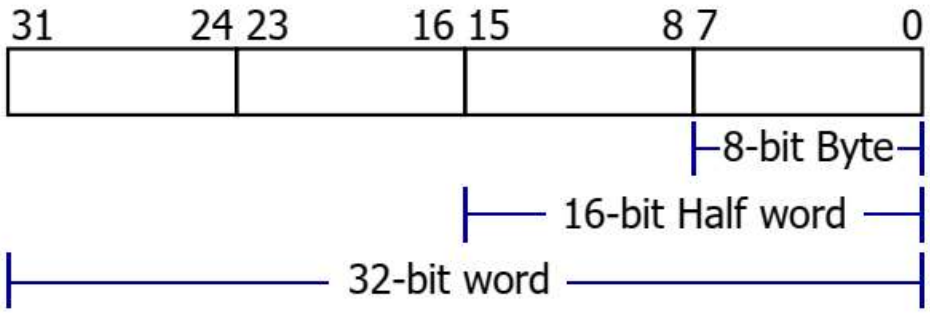
Thanh ghi

Only 16 registers are visible to a specific mode.A mode could access

– A particular set of r0-r12
– r13 (sp, stack pointer)
– r14 (lr, link register)
– r15 (pc, program counter)
– Current program status register (cpsr)
– The uses of r0-r13 are orthogonal

Cấu trúc ARM

General-purpose registers



- 6 data types (signed/unsigned)
- All ARM operations are 32-bit. Shorter data types are only supported by data transfer operations.

Cấu trúc ARM

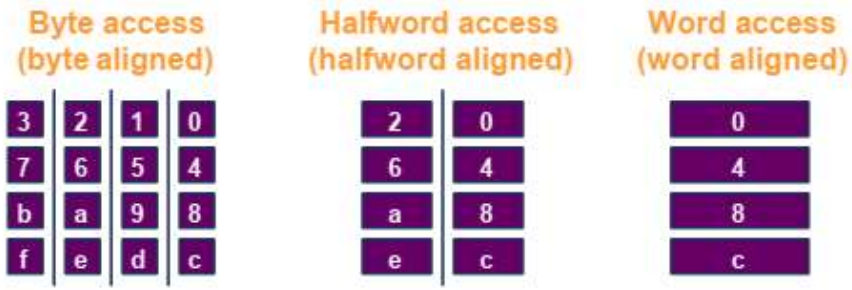
Program counter

- Store the address of the instruction to be executed
- All instructions are 32-bit wide and word-aligned
- Thus, the last two bits of pc are undefined.

Cấu trúc ARM

Data alignment

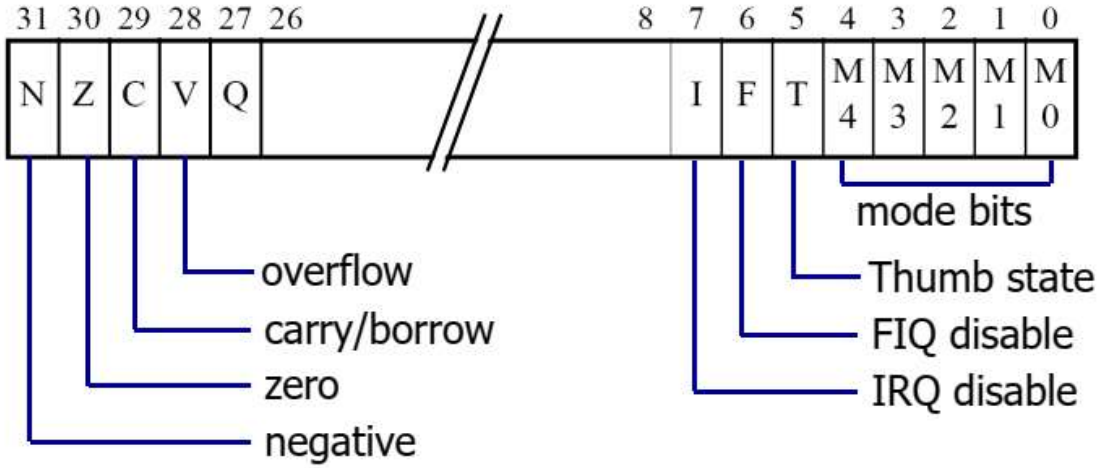
- Prior to architecture v6 data accesses must be appropriately aligned for access size
  - Unaligned addresses will produce unexpected/undefined results



- Unaligned data can be accessed using multiple aligned accesses combined with shift/mask operations

Cấu trúc ARM

Program status register (CPSR)

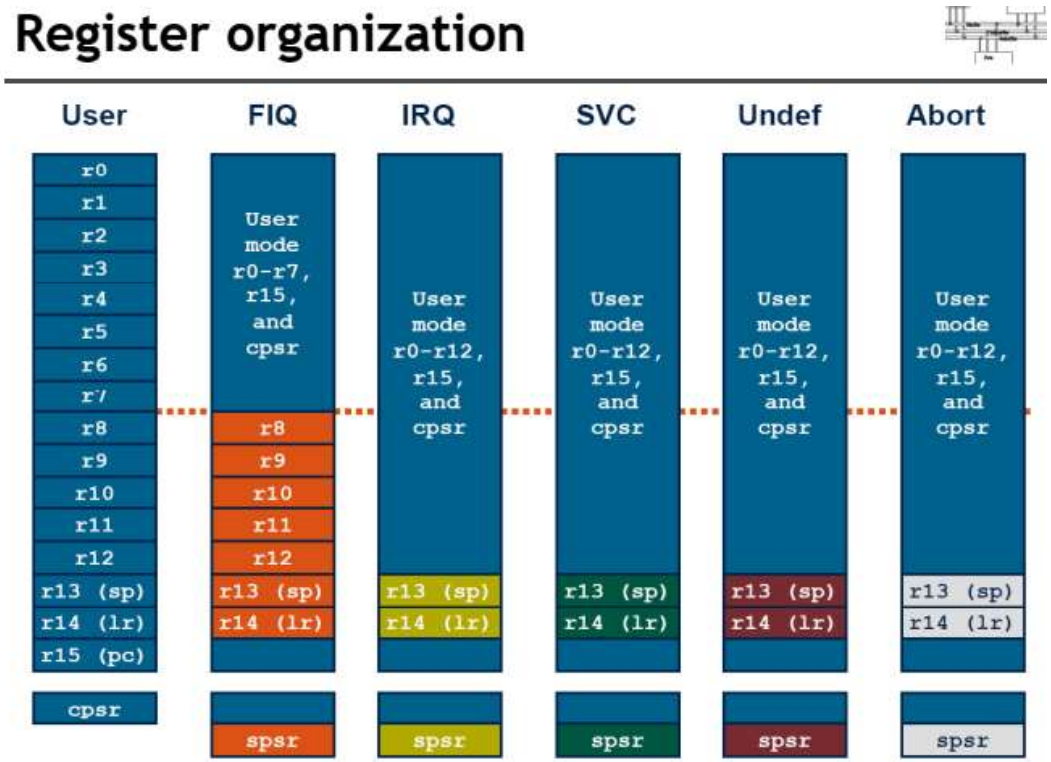


Cấu trúc ARM

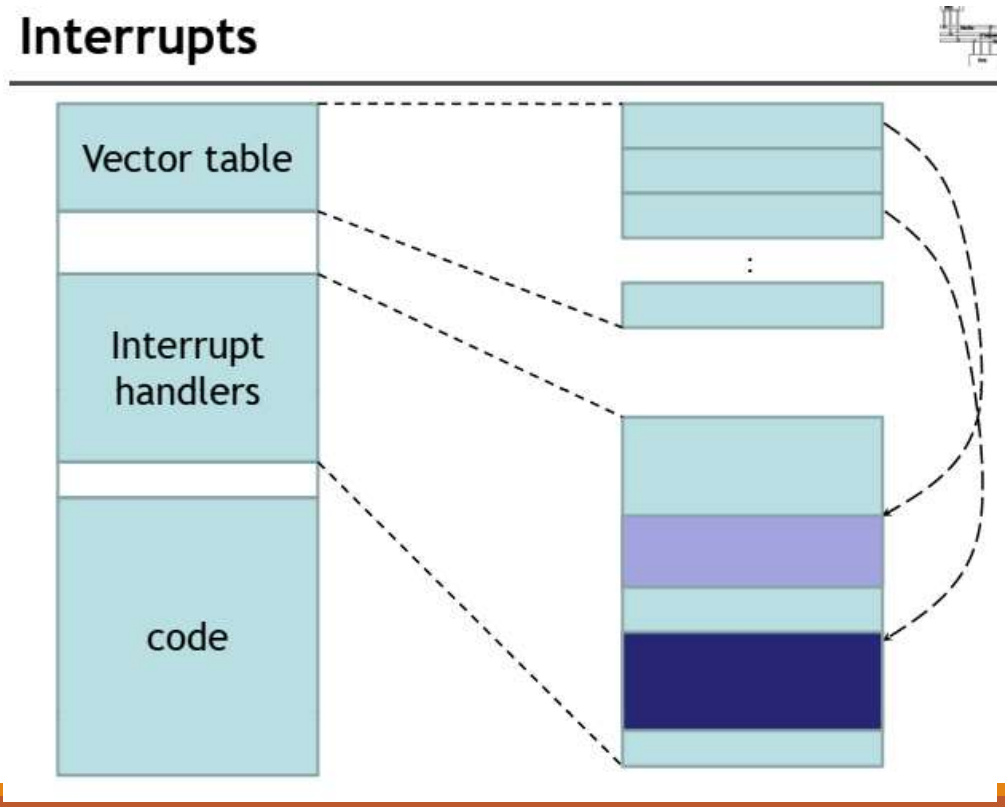
Processor modes

Processor mode		Description
User	usr	Normal program execution mode
FIQ	fiq	Supports a high-speed data transfer or channel process
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	A protected mode for the operating system
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs privileged operating system tasks

Cấu trúc ARM


Register organization
User
FIQ
IRQ
SVC
Undef
Abort
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
User mode r0-r7, r15, and cpsr
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr
User mode r0-r12, r15, and cpsr
r13 (sp)
r14 (lr)
spsr

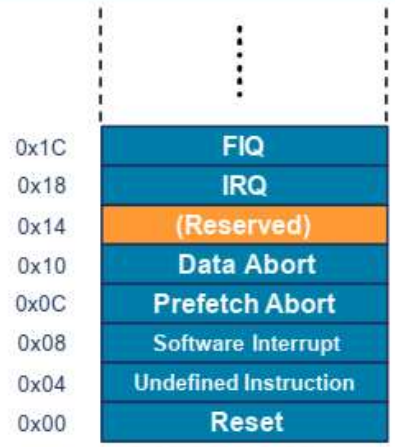
Cấu trúc ARM



Cấu trúc ARM

Exception Handling

- When an exception occurs, the core:
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - Change to ARM state
    - Change to exception mode
    - Disable interrupts (if appropriate)
  - Stores the return address in LR_<mode>
  - Sets PC to vector address
- To return, exception handler needs to:
  - Restore CPSR from SPSR_<mode>
  - Restore PC from LR_<mode>
- Must be done in ARM state in most cores, but...
  ...Thumb-2 capable cores can do this in Thumb state



Vector Table

Vector table can also be at
0xFFFF0000 on most cores

Tập lệnh ARM

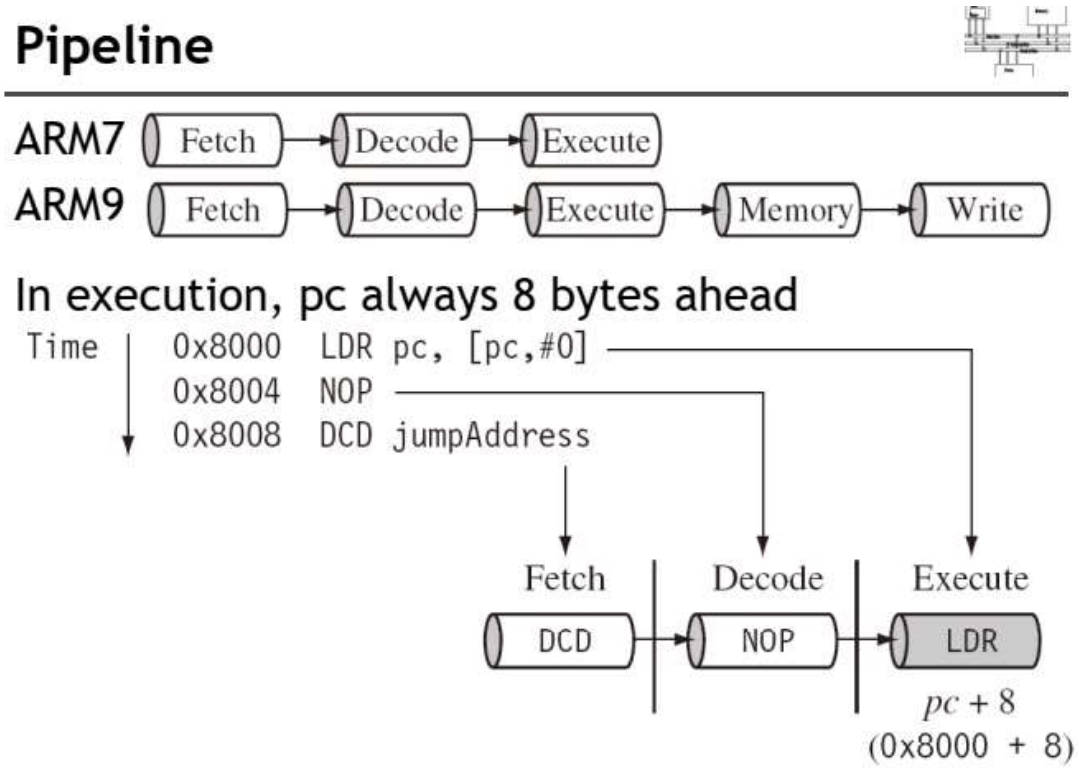
Instruction sets

- ARM/Thumb/Jazelle

	ARM (*cpsr* $T = 0$)	Thumb (*cpsr* $T = 1$)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution[a]	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers +*pc*	8 general-purpose registers +7 high registers +*pc*

	Jazelle (*cpsr* $T = 0, J = 1$)
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.

Tập lệnh ARM


Pipeline
ARM7
Fetch
Decode
Execute
ARM9
Fetch
Decode
Execute
Memory
Write
In execution, pc always 8 bytes ahead
Time
0x8000 LDR pc, [pc,#0]
0x8004 NOP
0x8008 DCD jumpAddress
Fetch
Decode
Execute
DCD
NOP
LDR
pc + 8
(0x8000 + 8)

Tập lệnh ARM

Việc thực hiện rẽ nhánh hoặc thay đổi PC làm cho Core ARM làm sạch pipeline

- ARM10 bắt đầu sử dụng dự đoán nhánh
- Một lệnh trong giai đoạn thực thi sẽ hoàn thành mặc dù một ngắt đã gọi. Các lệnh khác trong pipeline bị bỏ

Tập lệnh ARM

ARM Conditional Execution and Flags

- **ARM instructions can be made to execute conditionally by postfixing them with the appropriate condition code field.**
  - This improves code density *and* performance by reducing the number of forward branch instructions.

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

```
CMP    r3,#0
 ADDNE r0,r1,r2
```

- **By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". CMP does not need "S".**

```
loop
  _
  SUBS r1,r1,#1
  BNE loop
```

decrement r1 and set flags

if Z flag clear then branch

Tập lệnh ARM

ARM

Condition Codes

- **The possible condition codes are listed below:**
  - Note AL is the default and does not need to be specified

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

Tập lệnh ARM

ARM

Examples of conditional execution

- **Use a sequence of several conditional instructions**

```
if (a==0) func(1);
    CMP      r0,#0
    MOVEQ    r0,#1
    BLEQ     func
```

- **Set the flags, then use various condition codes**

```
if (a==0) x=0;
if (a>0)  x=1;
    CMP      r0,#0
    MOVEQ    r1,#0
    MOVGT    r1,#1
```

- **Use conditional compare instructions**

```
if (a==4 || a==10) x=0;
    CMP      r0,#4
    CMPNE    r0,#10
    MOVEQ    r1,#0
```

Tập lệnh ARM

Branch instructions

- **Branch :** `B{<cond>} label`
- **Branch with Link :** `BL{<cond>} subroutine_label`

- **The processor core shifts the offset field left by 2 positions, sign-extends it and adds it to the PC**
  - ± 32 Mbyte range
  - How to perform longer branches?

Tập lệnh ARM

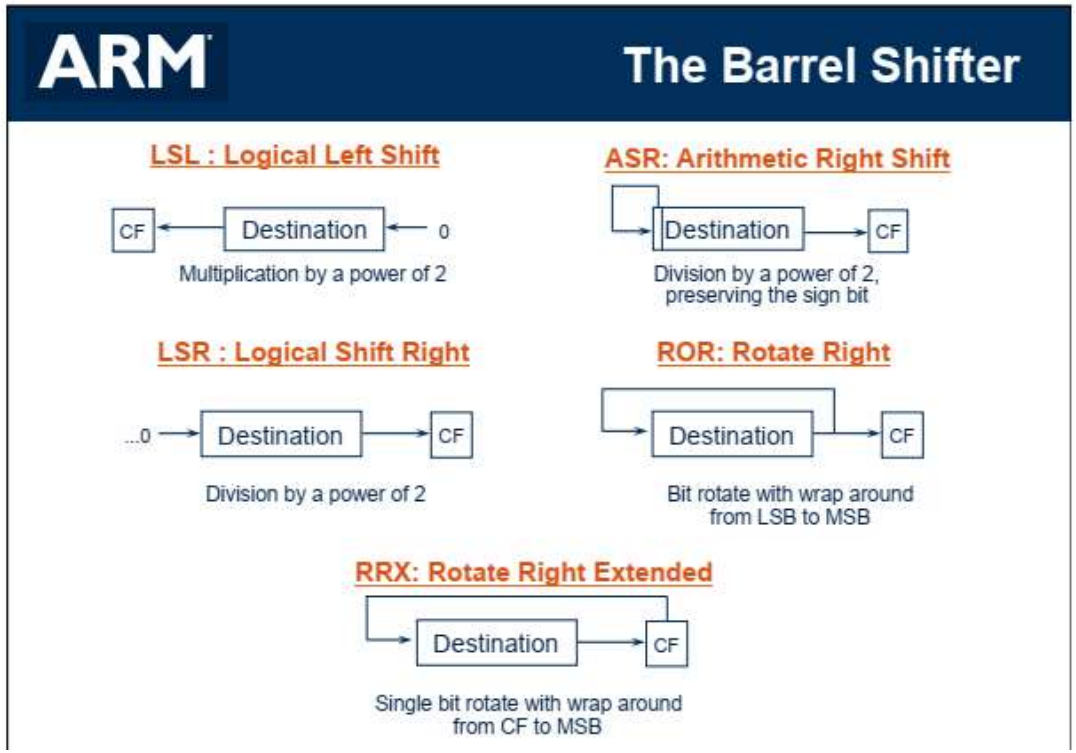
ARM Data processing Instructions

- **Consist of :**
  - Arithmetic: ADD ADC SUB SBC RSB RSC
  - Logical: AND ORR EOR BIC
  - Comparisons: CMP CMN TST TEQ
  - Data movement: MOV MVN
- **These instructions only work on registers, NOT memory.**
- **Syntax:**

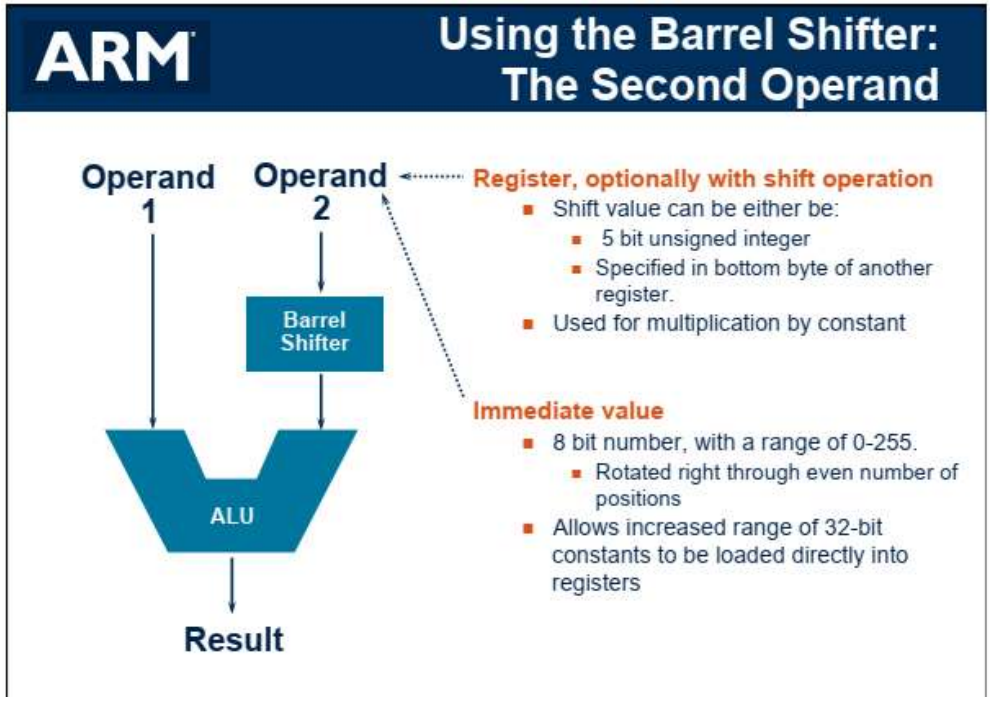
```
<Operation>{<cond>}{S} Rd, Rn, Operand2
```

  - Comparisons set flags only - they do not specify Rd
  - Data movement does not specify Rn
- **Second operand is sent to the ALU via barrel shifter.**

Tập lệnh ARM


ARM
The Barrel Shifter
LSL : Logical Left Shift
CF
Destination
0
Multiplication by a power of 2
ASR: Arithmetic Right Shift
Destination
CF
Division by a power of 2, preserving the sign bit
LSR : Logical Shift Right
...0
Destination
CF
Division by a power of 2
ROR: Rotate Right
Destination
CF
Bit rotate with wrap around from LSB to MSB
RRX: Rotate Right Extended
Destination
CF
Single bit rotate with wrap around from CF to MSB

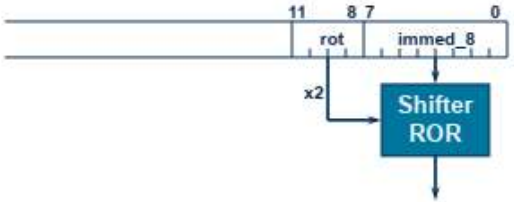
Tập lệnh ARM


ARM
Using the Barrel Shifter: The Second Operand
Operand 1
Operand 2
Barrel Shifter
ALU
Result
Register, optionally with shift operation
Shift value can be either be:
5 bit unsigned integer
Specified in bottom byte of another register.
Used for multiplication by constant
Immediate value
8 bit number, with a range of 0-255.
Rotated right through even number of positions
Allows increased range of 32-bit constants to be loaded directly into registers

Tập lệnh ARM

Immediate constants (1)

- **No ARM instruction can contain a 32 bit immediate constant**
  - All ARM instructions are fixed as 32 bits long
- **The data processing instruction format has 12 bits available for operand2**



Quick Quiz:

```
0xe3a004ff
MOV r0, #???
```

- **4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2**
- **Rule to remember is "8-bits shifted by an even number of bit positions".**

Tập lệnh ARM

Immediate constants (2)

- **Examples:**

- **The assembler converts immediate values to the rotate form:**
  - `MOV r0,#4096 ; uses 0x40 ror 26`
  - `ADD r1,r2,#0xFF0000 ; uses 0xFF ror 16`
- **The bitwise complements can also be formed using MVN:**
  - `MOV r0, #0xFFFFFFFF ; assembles to MVN r0,#0`
- **Values that cannot be generated in this way will cause an error.**

Tập lệnh ARM

ARM

Loading 32 bit constants

- **To allow larger constants to be loaded, the assembler offers a pseudo-instruction:**
  - `LDR rd, =const`
- **This will either:**
  - Produce a `MOV` or `MVN` instruction to generate the value (if possible).

  or
  - Generate a `LDR` instruction with a PC-relative address to read the constant from a *literal pool* (Constant data area embedded in the code).
- **For example**
  - `LDR r0,=0xFF` => `MOV r0,#0xFF`
  - `LDR r0,=0x55555555` => `LDR r0,[PC,#Imm12]`

```
...
...
DCD 0x55555555
```

- **This is the recommended way of loading constants into a register**

Tập lệnh ARM

ARM — Multiply

- **Syntax:**
  - MUL{<cond>}{S} Rd, Rm, Rs — Rd = Rm * Rs
  - MLA{<cond>}{S} Rd,Rm,Rs,Rn — Rd = (Rm * Rs) + Rn
  - [U|S]MULL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := Rm*Rs
  - [U|S]MLAL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := (Rm*Rs)+RdHi,RdLo
- **Cycle time**
  - Basic MUL instruction
    - 2-5 cycles on ARM7TDMI
    - 1-3 cycles on StrongARM/XScale
    - 2 cycles on ARM9E/ARM102xE
  - +1 cycle for ARM9TDMI (over ARM7TDMI)
  - +1 cycle for accumulate (not on 9E though result delay is one cycle longer)
  - +1 cycle for "long"
- **Above are "general rules" - refer to the TRM for the core you are using for the exact details**

Tập lệnh ARM

ARM — Single register data transfer

LDR	STR	Word
LDRB	STRB	Byte
LDRH	STRH	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load

- **Memory system must support all access sizes**

- **Syntax:**
  - LDR{<cond>}{<size>} Rd, <address>
  - STR{<cond>}{<size>} Rd, <address>

  e.g. LDREQB

Tập lệnh ARM

ARM

Address accessed

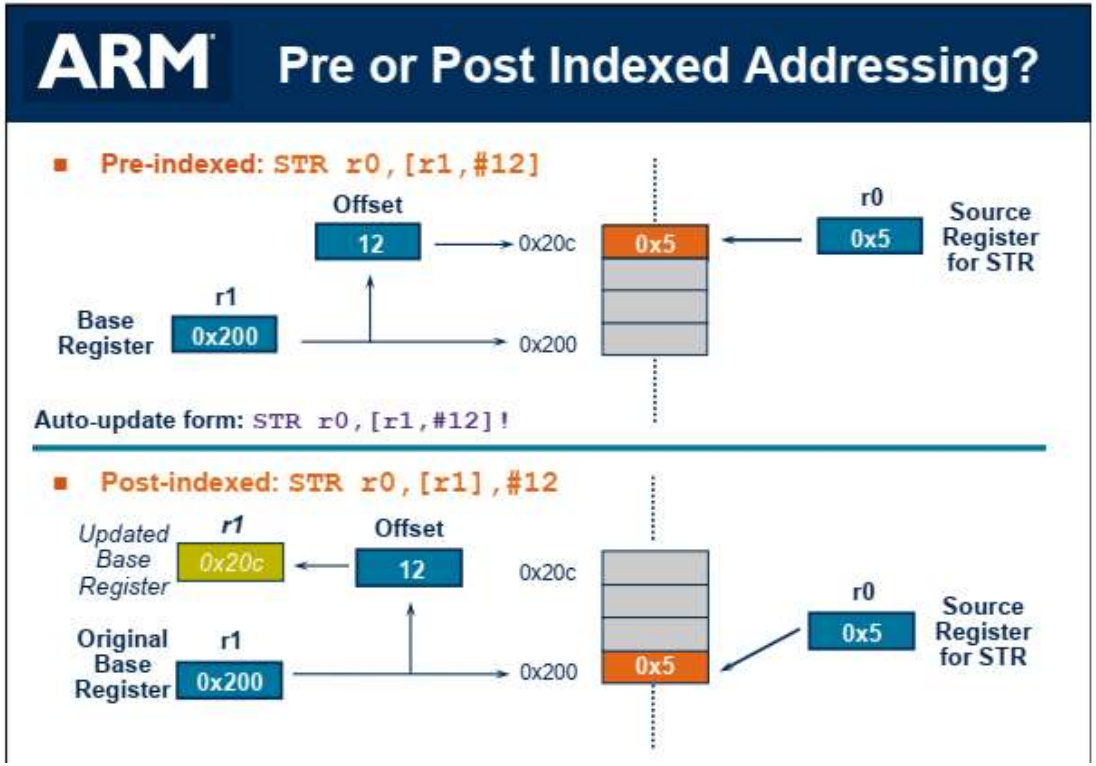
- **Address accessed by LDR/STR is specified by a base register plus an offset**
- **For word and unsigned byte accesses, offset can be**
  - An unsigned 12-bit immediate value (ie 0 - 4095 bytes).

    ```
    LDR r0,[r1,#8]
    ```
  - A register, optionally shifted by an immediate value

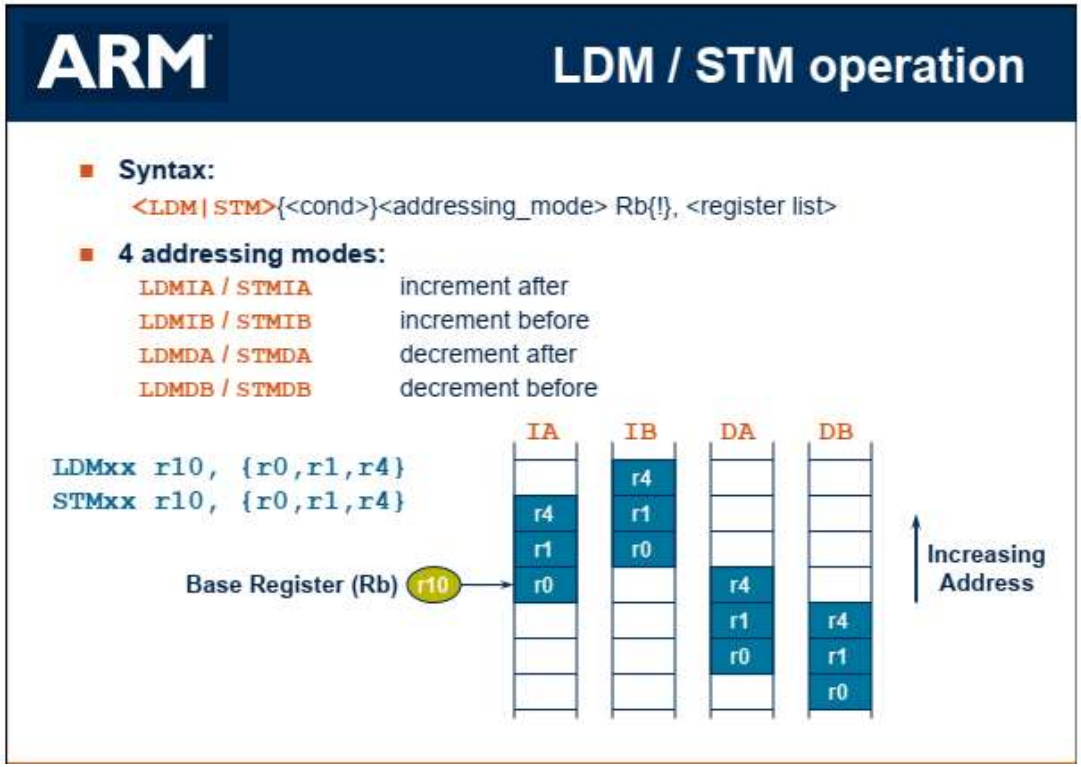
    ```
    LDR r0,[r1,r2]
    LDR r0,[r1,r2,LSL#2]
    ```
- **This can be either added or subtracted from the base register:**

  ```
  LDR r0,[r1,#-8]
  LDR r0,[r1,-r2]
  LDR r0,[r1,-r2,LSL#2]
  ```
- **For halfword and signed halfword / byte, offset can be:**
  - An unsigned 8 bit immediate value (ie 0-255 bytes).
  - A register (unshifted).
- **Choice of *pre-indexed* or *post-indexed* addressing**

Tập lệnh ARM


ARM
Pre or Post Indexed Addressing?
Pre-indexed: STR r0,[r1,#12]
Offset
12
0x20c
0x5
r0
0x5
Source Register for STR
r1
Base Register
0x200
0x200
Auto-update form: STR r0,[r1,#12]!
Post-indexed: STR r0,[r1],#12
Updated Base Register
r1
0x20c
Offset
12
0x20c
r0
0x5
Source Register for STR
Original Base Register
r1
0x200
0x200
0x5

Tập lệnh ARM


ARM
LDM / STM operation
Syntax:
<LDM | STM>{<cond>}<addressing_mode> Rb{!}, <register list>
4 addressing modes:
LDMIA / STMIA increment after
LDMIB / STMIB increment before
LDMDA / STMDA decrement after
LDMDB / STMDB decrement before
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
Base Register (Rb)
r10
IA
IB
DA
DB
r4
r1
r0
Increasing Address

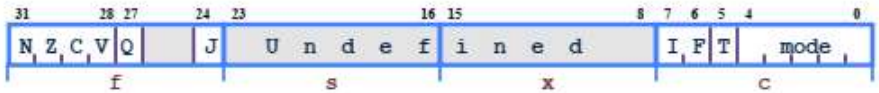
Tập lệnh ARM

Software Interrupt (SWI)

- Causes an exception trap to the SWI hardware vector
- The SWI handler can examine the SWI number to decide what operation has been requested.
- By using the SWI mechanism, an operating system can implement a set of privileged operations which applications running in user mode can request.
- Syntax:
  - `SWI{<cond>} <SWI number>`

Tập lệnh ARM

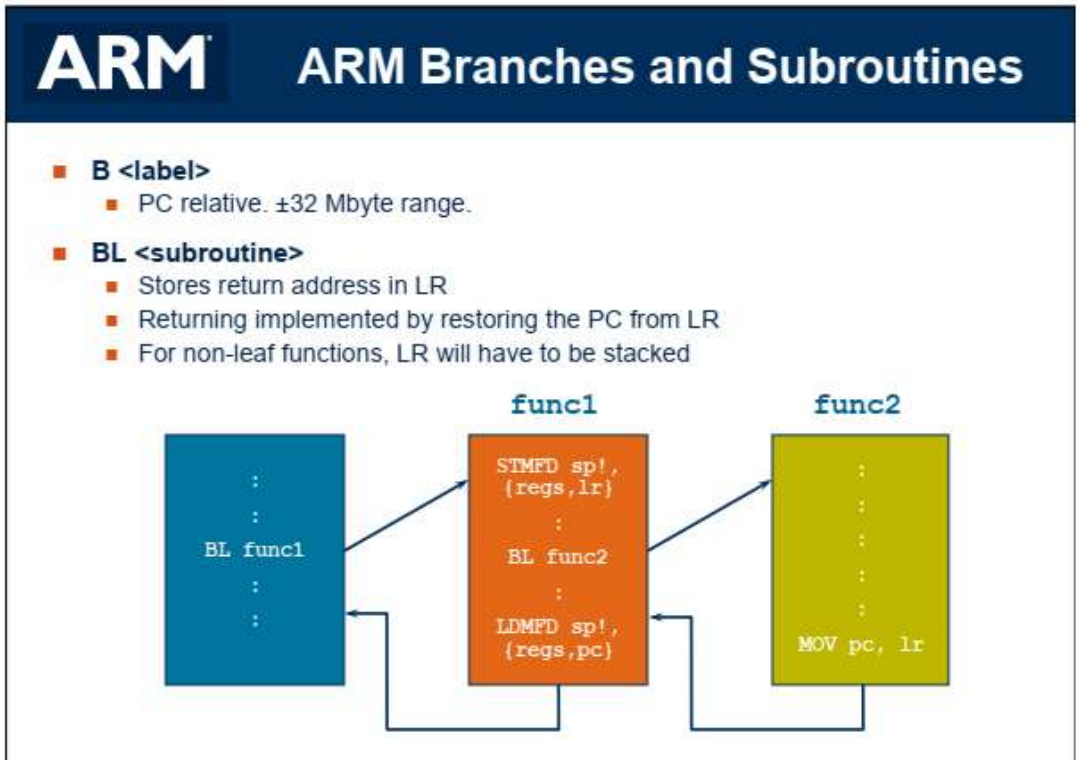
ARM — PSR Transfer Instructions



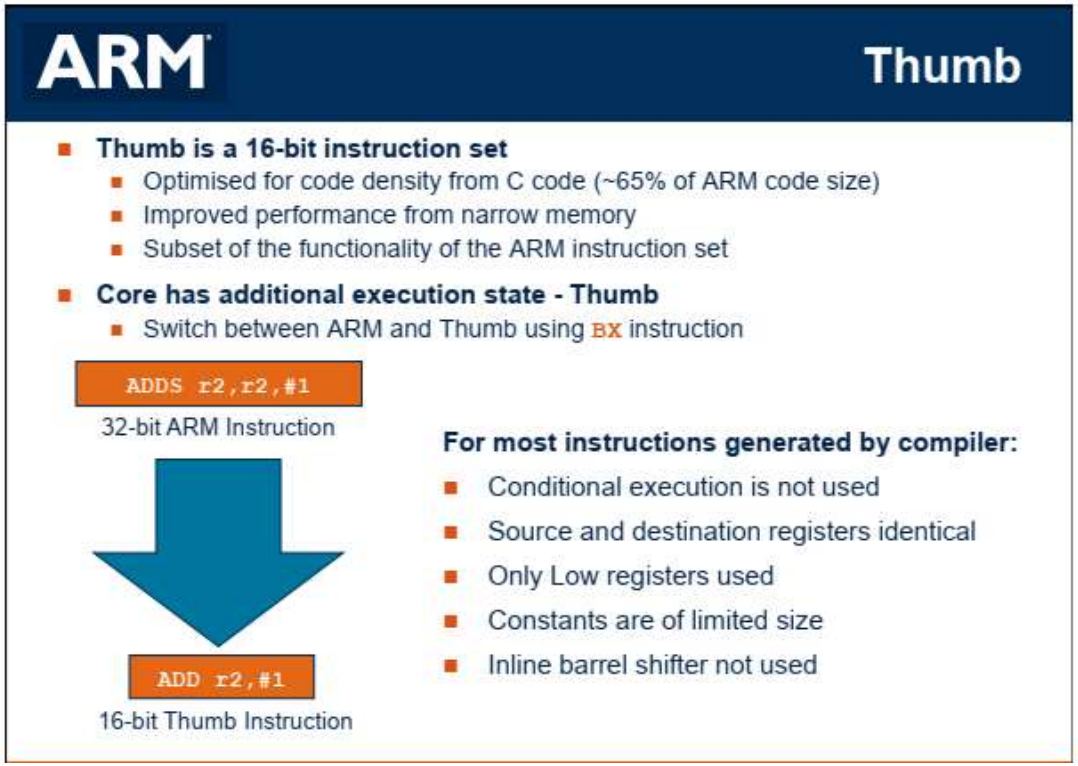
- MRS and MSR allow contents of CPSR / SPSR to be transferred to / from a general purpose register.
- Syntax:
  - `MRS{<cond>} Rd,<psr>  ; Rd = <psr>`
  - `MSR{<cond>} <psr[_fields]>,Rm ; <psr[_fields]> = Rm`

where
  - `<psr> = CPSR or SPSR`
  - `[_fields] = any combination of 'fsxc'`
- Also an immediate form
  - `MSR{<cond>} <psr_fields>,#Immediate`
- In User Mode, all bits can be read but only the condition flags (_f) can be written.

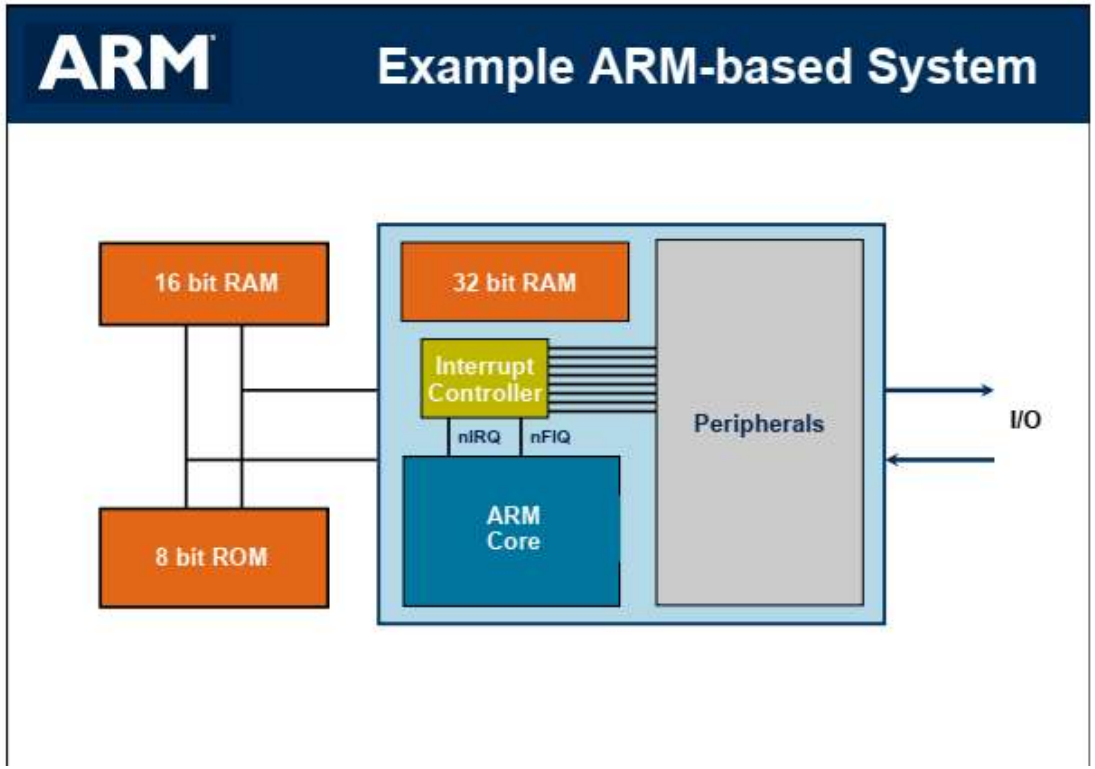
Tập lệnh ARM


ARM
ARM Branches and Subroutines
B <label>
PC relative. ±32 Mbyte range.
BL <subroutine>
Stores return address in LR
Returning implemented by restoring the PC from LR
For non-leaf functions, LR will have to be stacked
func1
func2
BL func1
STMFD sp!, {regs,lr}
BL func2
LDMFD sp!, {regs,pc}
MOV pc, lr

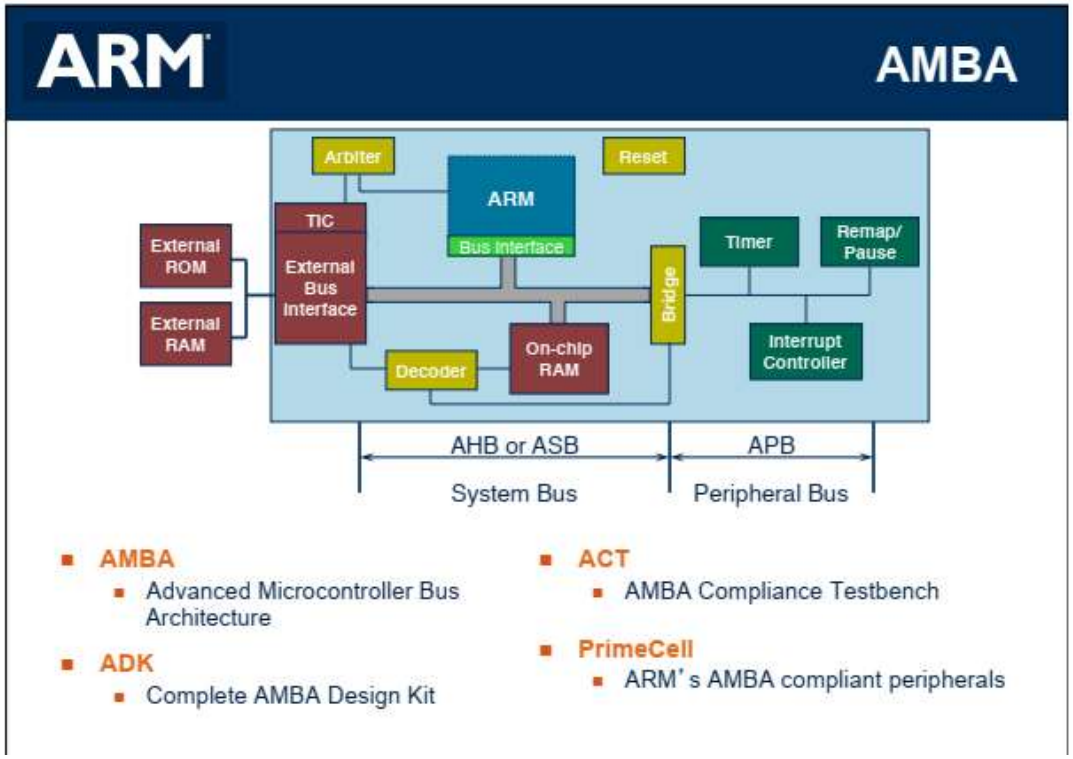
Tập lệnh ARM


ARM
Thumb
Thumb is a 16-bit instruction set
Optimised for code density from C code (~65% of ARM code size)
Improved performance from narrow memory
Subset of the functionality of the ARM instruction set
Core has additional execution state - Thumb
Switch between ARM and Thumb using BX instruction
ADDS r2,r2,#1
32-bit ARM Instruction
ADD r2,#1
16-bit Thumb Instruction
For most instructions generated by compiler:
Conditional execution is not used
Source and destination registers identical
Only Low registers used
Constants are of limited size
Inline barrel shifter not used

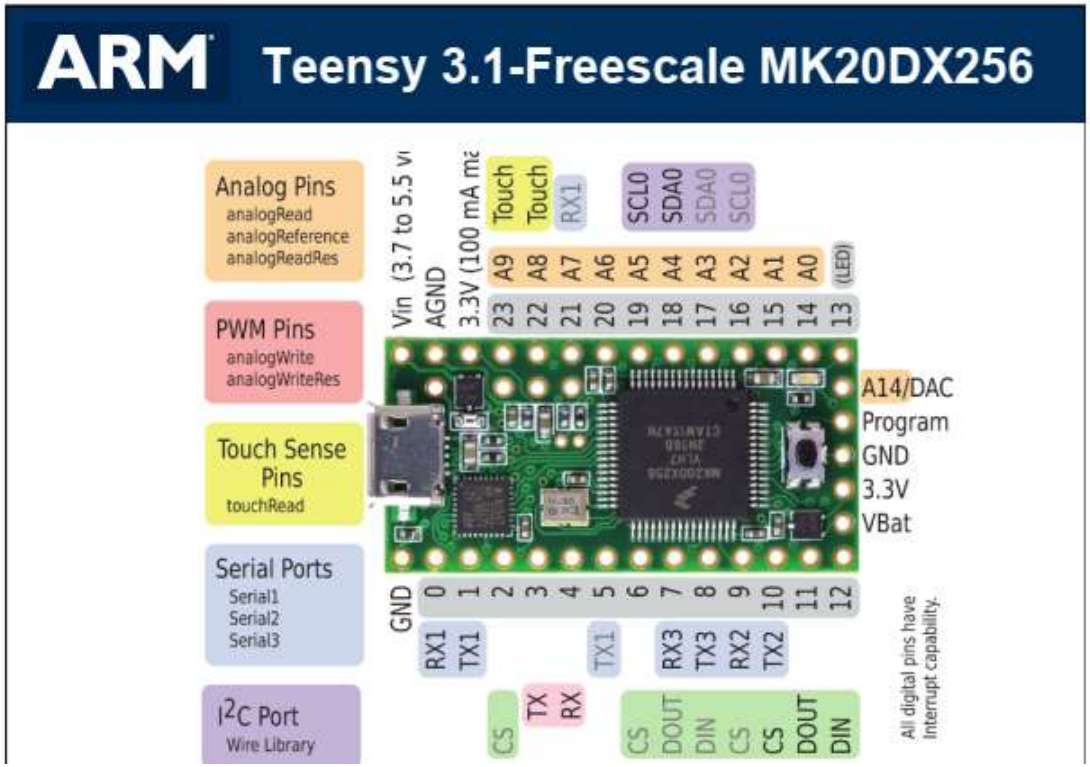
System design



System design

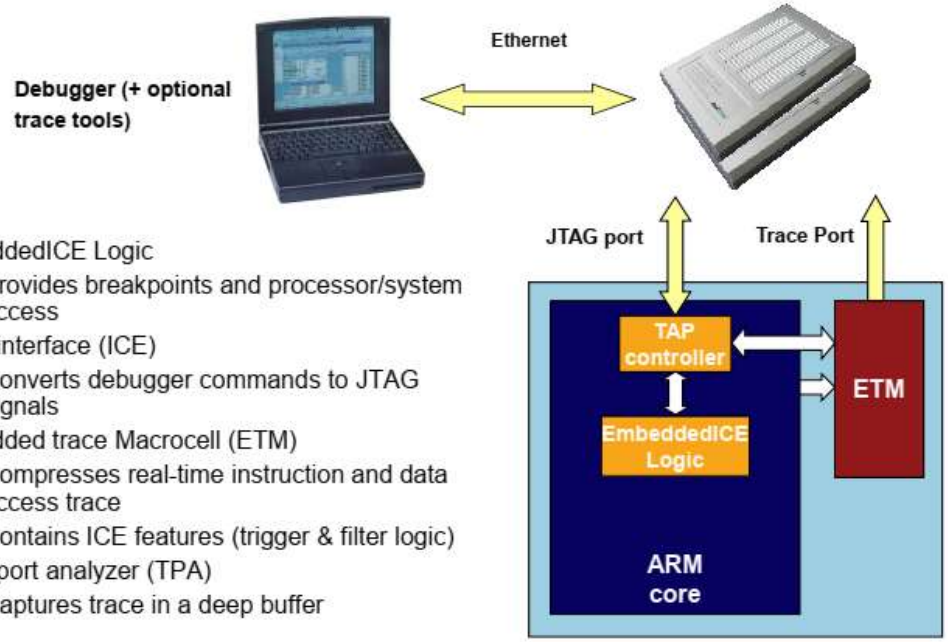


System design



Tools

ARM Debug Architecture



- EmbeddedICE Logic
  - Provides breakpoints and processor/system access
- JTAG interface (ICE)
  - Converts debugger commands to JTAG signals
- Embedded trace Macrocell (ETM)
  - Compresses real-time instruction and data access trace
  - Contains ICE features (trigger & filter logic)
- Trace port analyzer (TPA)
  - Captures trace in a deep buffer

Tools

Keil Development Tools for ARM

- Includes ARM macro assembler, compilers (ARM RealView C/C++ Compiler, Keil CARM Compiler, or GNU compiler), ARM linker, Keil uVision Debugger and Keil uVision IDE
- Keil uVision Debugger accurately simulates on-chip peripherals (I^2C, CAN, UART, SPI, Interrupts, I/O Ports, A/D and D/A converters, PWM, etc.)
- Evaluation Limitations
  - 16K byte object code + 16K data limitation
  - Some linker restrictions such as base addresses for code/constants
  - GNU tools provided are not restricted in any way
- http://www.keil.com/demo/