Table of Contents

ISA

Major elements of an Instruction Set Architecture

- word size
- registers
- memory
- endianess
- conditions
- instructions
- addressing modes

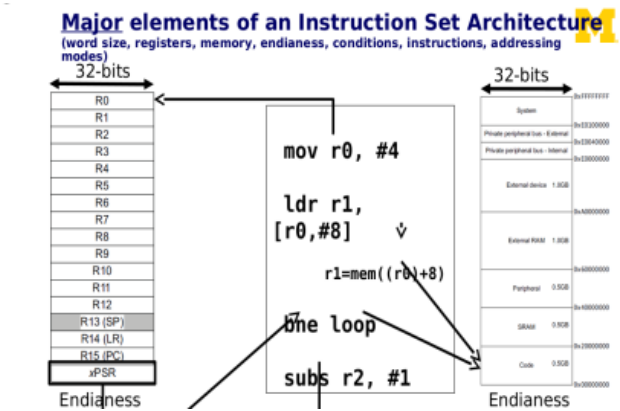


Figure 1

Cortex M4 Program Image

The program image in Cortex-M4 contains

- Vector table: includes the starting addresses of exceptions (vectors) and the value of the main stack point (MSP);
- C start-up routine;
- Program code: application code and data;
- C library code: program codes for C library functions

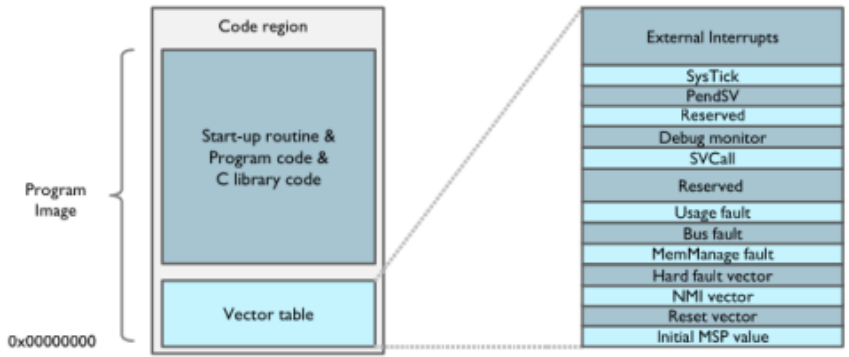


Figure 2

Cortex M4 Program Image

After Reset, the processor

- First reads the initial MSP value;
- Then reads the reset vector;
- Branches to the start of the program execution address (reset handler);
- Subsequently executes program instructions

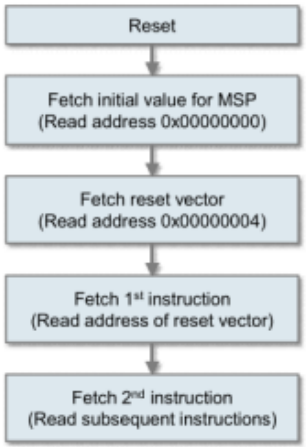
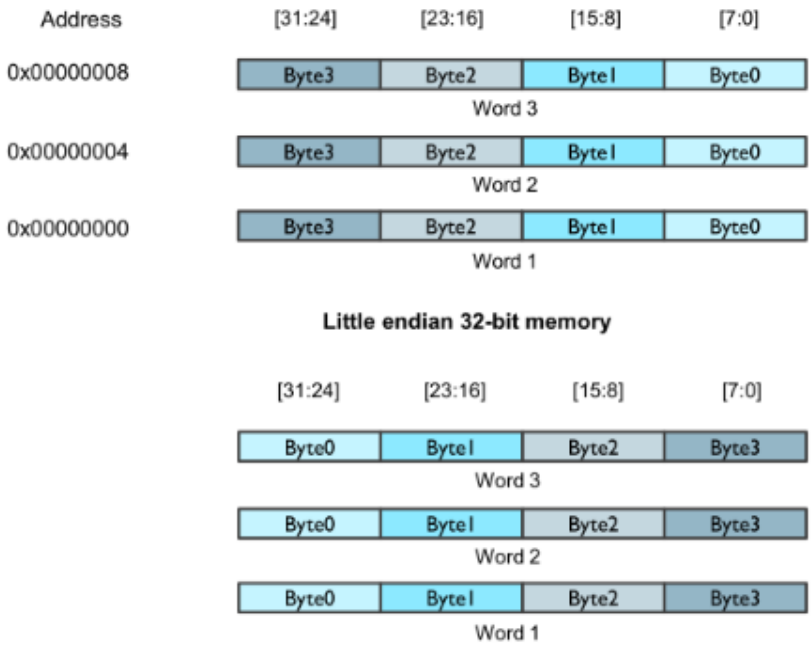


Figure 3

Cortex-M4 Endianness

- Endian refers to the order of bytes stored in memory
  - Little endian: lowest byte of a word-size data is stored in bit 0 to bit 7
  - Big endian: lowest byte of a word-size data is stored in bit 24 to bit 31
- Cortex-M4 supports both little endian and big endian
- However, Endianness only exists in the hardware level



Main Features of Cortex M4 Instruction Set

- All instructions are 32 bits long.
- Supports 32-bit Thumb-2 instructions
- Possible to handle all processing requirements in one operation state (Thumb state)
- No need to separate ARM code and Thumb code source files, which makes the development and maintenance of software easier
- Most instructions execute in a single cycle.
- Every instruction can be conditionally executed.
- A load/store architecture
-
  - Data processing instructions act only on registers
    - Three operand format
    - Combined ALU and shifter for high speed bit manipulation
  - Specific memory access instructions with powerful auto-indexing addressing modes.
    - 32 bit and 8 bit data types
    - Flexible multiple register load and store instructions

Main Features of Cortex M4 Instruction Set

- Standard Instruction Format for ARM Assembly program:
  label
  mnemonic operand1, operand2, ...; Comments
- label: used as a reference to an address location (optional)
- mnemonic: name of the instruction
- operand: can be an ARM Cortex M4 register, a constant, or another instruction-specific parameter.
  - For data processing instruction, first operand is the destination (destination register) of the operation
  - For memory read instruction (except multiple load instruction), first operand is the register which data is loaded into
  - For memory write instruction (except multiple store instruction), first operand is the register that holds the data to be written to the memory
- Second operand can be either be a register or a constant
  - MOV Rm, "0x22"; Set Rm= 0x22 (hexadecimal)
  - ANDS R0,R1 ; R0=R0 AND R1, update APSR

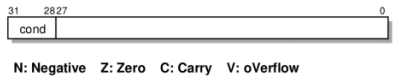
Conditional Execution

- 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
- The conditional execution code is faster and smaller
- 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

```
; 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
```

Figure 5



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

Figure 7

Commonly Used Directives

Table 1: Commonly Used Directives for Inserting Data into a Program

Type of Data to Insert	ARM Assembler
Byte	DCB (example:DCB 0x12)
Half-word	DCW (example: DCW 0x1234)
Word	DCD (example: DCW 0x12345678)
String	DCB (example: "hello" 0)

Figure 8

Commonly Used Directives

Directive	**ARM Assembler**
AREA	Instruct the assembler to assemble a new code or data section
SPACE	Reserve a block of memory and fills it with zero
FILL	Reserve a block of memory and fills it with the specified value. The size of the value can be byte, half-word or word
ALIGN	Align the current location to a specified boundary by padding with zeros. Example: ALIGN 8: make sure the next instruction or data is aligned to 8 byte boundary

Commonly Used Directives

Table 5.3 Suffixes for Cortex-M Assembly Language

Suffixes	Descriptions
S	Update APSR (Application Program Status Register, such as Carry, Overflow, Zero and Negative flags); for example: ADDS R0, R1 ; this ADD operation will update APSR

EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE	Conditional execution. EQ = Equal, NE = Not Equal, LT = Less Than, GT = Greater Than, etc. On the Cortex-M processors these conditions can be applied to conditional branches; for example:
	`BEQ label ; Branch to label if previous operation result in`
	`; equal status`
	or conditionally executed instructions (see IF-THEN instruction in section 5.6.9); for example:
	`ADDEQ R0, R1, R2 ; Carry out the add operation if the previous`
	`; operation results in equal status`

Figure 9

Conditional Execution: Example

- Example: Explain the use of conditional instructions to update the value of R4 if the signed values R0 is greater than R1 and R2 is greater than R3.

```
CMP     R0, R1      ; Compare R0 and R1, setting flags
ITT     GT          ; Skip next two instructions unless GT condition holds
CMPGT   R2, R3      ; If 'greater than', compare R2 and R3, setting flags
MOVGT   R4, R5      ; If still 'greater than', do R4 = R5
```

Figure 10

Instruction Set

The instructions in the Cortex-M4 processors can be divided into various groups based on functionality:

- Moving data within the processor
- Memory accesses
- Arithmetic operations
- Logic operations
- Shift and Rotate operations
- Conversion (extend and reverse ordering) operations
- Bit field processing instructions
- Program flow control (branch, conditional branch, conditional execution, and function calls)
- Multiply accumulate (MAC) instructions
- Divide instructions
- Memory barrier instructions
- Exception-related instructions
- Sleep mode-related instructions
- Other functions
- In addition, the Cortex-M4 processor supports the Enhanced DSP instructions, SIMD operations and packing instructions, Adding fast multiply and MAC instructions and Floating point instructions (if the floating point unit is present)

Moving Data within the processor

Table 5.4 Instructions for Transferring Data within the Processor

Instruction	Dest	Source	Operations
MOV	R4,	R0	; Copy value from R0 to R4
MOVS	R4,	R0	; Copy value from R0 to R4 with APSR (flags) update
MRS	R7,	PRIMASK	; Copy value of PRIMASK (special register) to R7
MSR	CONTROL,	R2	; Copy value of R2 into CONTROL (special register)
MOV	R3,	#0x34	; Set R3 value to 0x34
MOVS	R3,	#0x34	; Set R3 value to 0x34 with APSR update
MOVW	R6,	#0x1234	; Set R6 to a 16-bit constant 0x1234
MOVT	R6,	#0x8765	; Set the upper 16-bit of R6 to 0x8765
MVN	R3,	R7	; Move negative value of R7 into R3

Figure 11

Memory Access Instruction

Table 5.6 Memory Access Instructions for Various Data Sizes

Data Type	Load (Read from Memory)	Store (Write to Memory)
8-bit unsigned	LDRB	STRB
8-bit signed	LDRSB	STRB
16-bit unsigned	LDRH	STRH
16-bit signed	LDRSH	STRH
32-bit	LDR	STR
Multiple 32-bit	LDM	STM
Double-word (64-bit)	LDRD	STRD
Stack operations (32-bit)	POP	PUSH

Figure 12

Cortex M4 Instruction Cycle

- STR Rx,[Ry,#imm] is always one cycle
- LDR PC,[any] is always a blocking operation. This means at least two cycles for the load, and three cycles for the pipeline reload. So this operation takes at least five cycles, or more if stalled on the load or the fetch.
- Any load or store that generates an address dependent on the result of a preceding data processing operation stalls the pipeline for an additional cycle while the register bank is updated.
- LDM and STM cannot be pipelined with preceding or following instructions. However, all elements after the first are pipelined together. So, a three element LDM takes 2+1+1 or 5^5 cycles when not stalled. Similarly, an eight-element store takes nine cycles when not stalled. When interrupted, LDM and STM instructions continue from where they left off when returned to. The continue operation adds one or two cycles to the first element when started.

Addressing Modes

- CPU determines the memory address by adding a positive or negative offset to the value in a base register.
- The way in which the CPU combines these two parts is called the Addressing Mode.
- The ARM instruction set architecture has the following addressing modes:
- **Immediate Addressing Mode**
  - The offset is an unsigned integer that is stored as part of the instruction.
  - It can be added to or subtracted from the value in the base register.
  - If a label is used to specify the address, the assembler uses the PC (Program Counter) as the base register and computes the appropriate offset.
  - MOV R0, #x ; Set R0 = x
  - ADD R1, R2, #12; R1=R2+12

Addressing Modes

- **Register Addressing Mode**
  - The offset is an unsigned integer that is in a register other than the PC.
  - Direct: MOV R0, R1 ; Set R0=R1
  - Indirect: LDR R0, [R1] ; Load R0 with the word pointed by R1
- **Pre-Indexed Addressing Mode**
  - Register indirect with offset: LDR R0, [R1, #x]
    - Effective address: R1+x
    - Loads R0 with the word pointed at by R1+x
  - Register indirect pre-incrementing: LDR R0, [R1, #x]!
    - Effective address: R1+x
    - Loads R0 with the word pointed at by R1+x then update the pointer by adding x to R1

Addressing Modes

- **Post-Indexing Addressing Mode** : LDR R0, [R1], #x
  - Effective address: R1
  - Loads R0 with the word pointed at by R1 then update the pointer by adding x to R1
- **Program Counter Relative (PC Relative) Addressing Mode**
  - LDR R0, [R15, #24]
  - Loads R0 with the word pointed at by R15+24

Addressing Modes

Addressing Mode	Assembly Mnemonic	Effective address	FinalValue in R1
Pre-indexed, base (unchanged)	LDR R0, [R1, #x]	R1 + x	R1
Pre-indexed, base (updated)	LDR R0, [R1, #x]!	R1 + x	R1+x
Post-indexed, base (updated)	LDR R0, [R1], #x	R1	R1+x

Immediate Addressing Mode

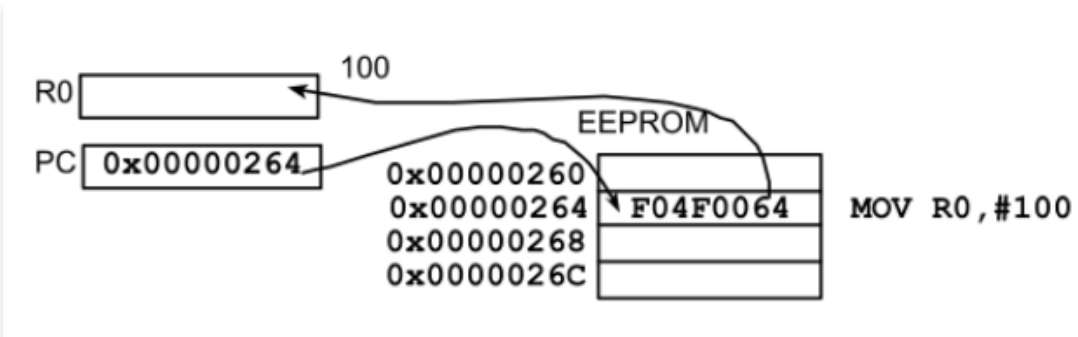


Figure 13

Register Indirect Addressing Mode

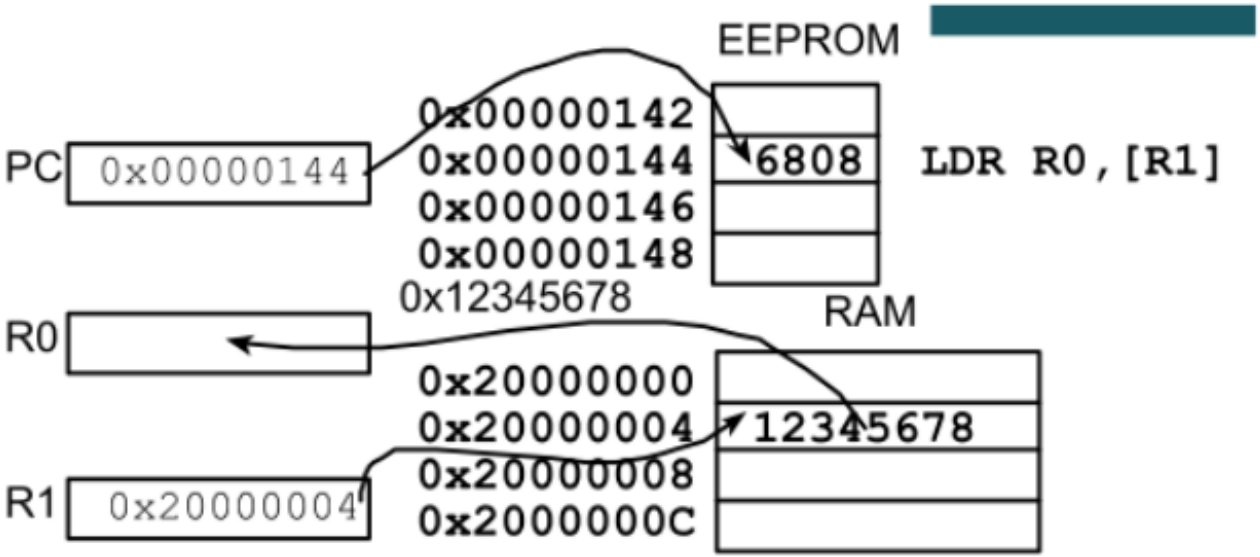


Figure 14

Pre-Indexed Addressing Mode

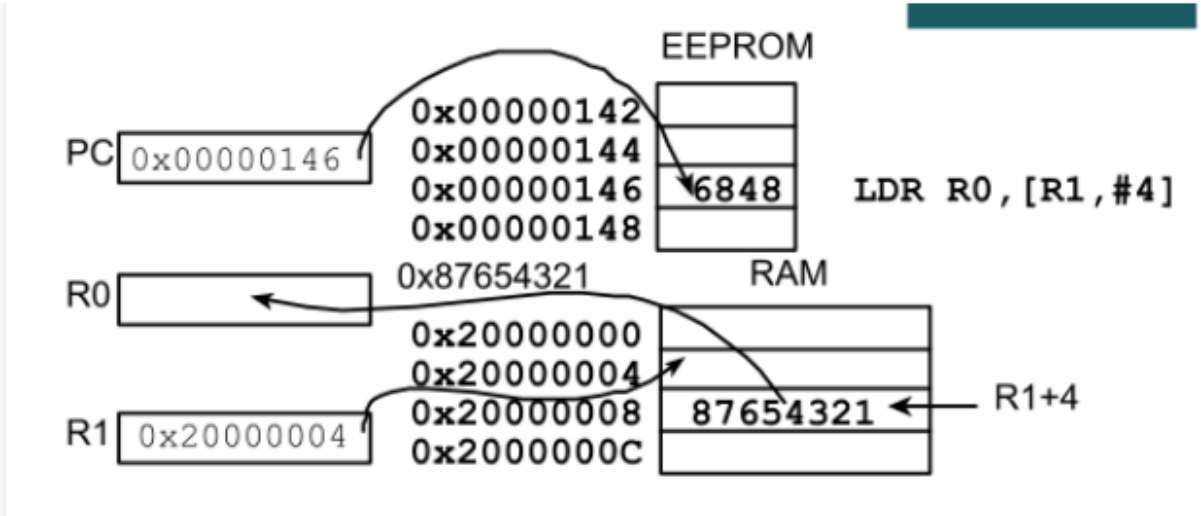


Figure 15