Overview of Computer Architecture

Processors and Execution

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

November 8, 2019

http://carlhenrik.com

Computer Science

Definition (Computer Science)

The systematic study (science) of algorithms

Definition (Algorithms)

an ordered set of unambigous, executable steps that defines a terminating process

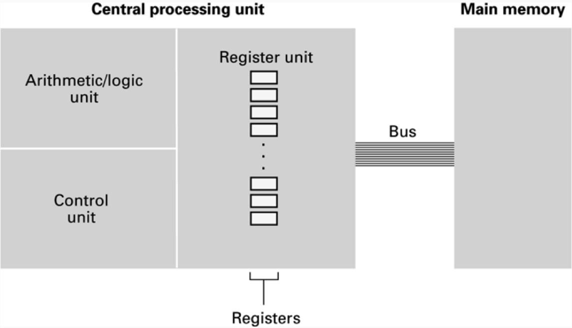
Computing

Computer Architecture

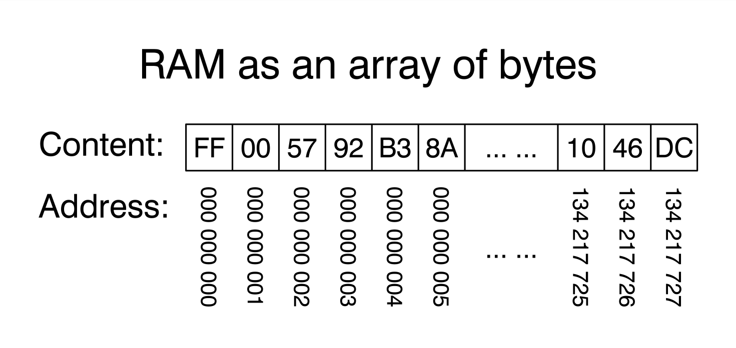
Execution

Start

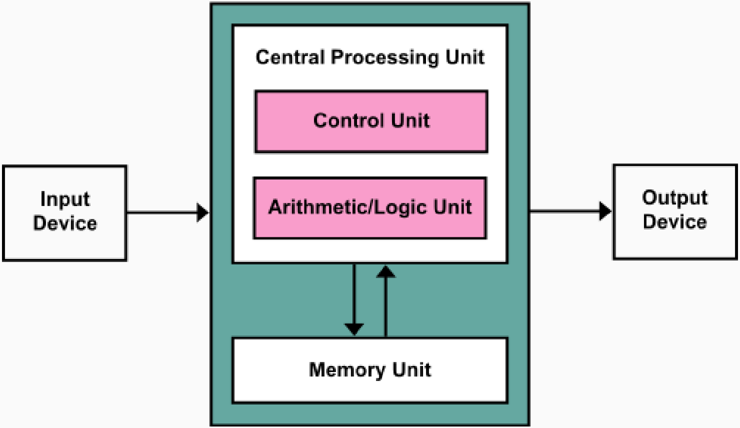
Computer Architecture



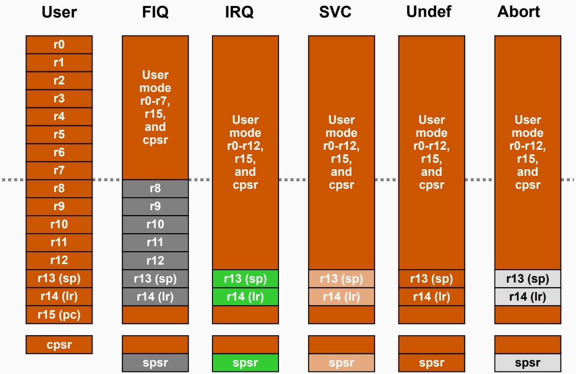
Memory


RAM as an array of bytes
Content:
FF
00
57
92
B3
8A
... ...
10
46
DC
Address:
000 000 000
000 000 001
000 000 002
000 000 003
000 000 004
000 000 005
... ...
134 217 725
134 217 726
134 217 727

CPU


Central Processing Unit
Control Unit
Arithmetic/Logic Unit
Input
Device
Output
Device
Memory Unit

Register


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

Program Counter

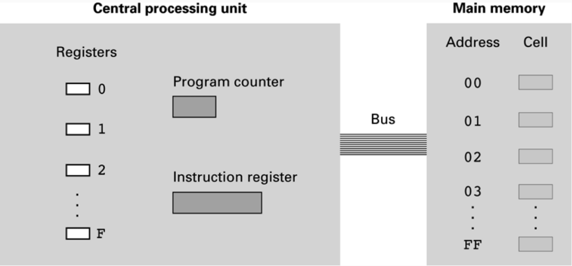
Program Counter

- Address to memory to fetch instruction to execute
- Start position

Instruction Register

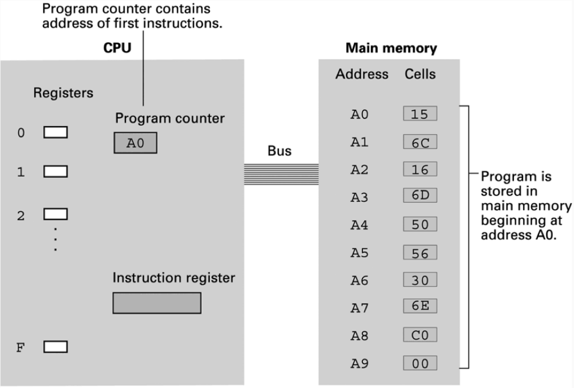
- Stores instruction

Execution



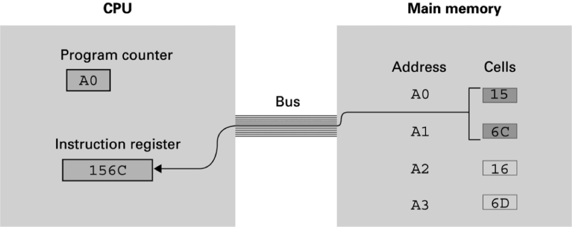
Central processing unit
Main memory
Registers
0
1
2
F
Program counter
Instruction register
Bus
Address
Cell
00
01
02
03
FF

Execution



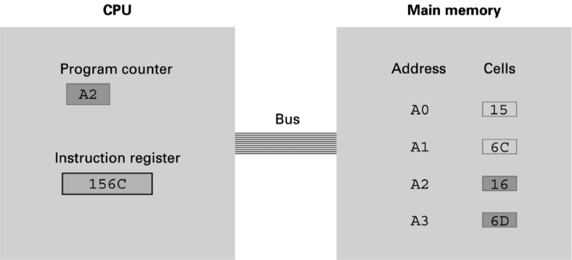
Program counter contains
address of first instructions.
CPU
Main memory
Registers
0
1
2
F
Program counter
A0
Instruction register
Bus
Address
Cells
A0
15
A1
6C
A2
16
A3
6D
A4
50
A5
56
A6
30
A7
6E
A8
C0
A9
00
Program is
stored in
main memory
beginning at
address A0.

Execution



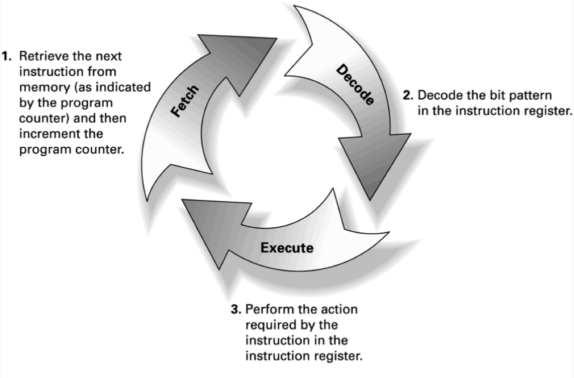
a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Execution



b. Then the program counter is incremented so that it points to the next instruction.

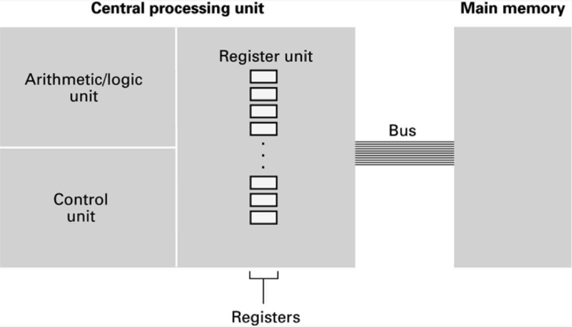
Execution



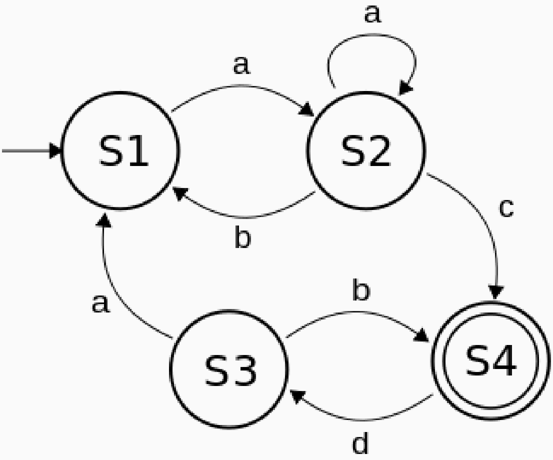
1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.
Fetch
Decode
2. Decode the bit pattern in the instruction register.
Execute
3. Perform the action required by the instruction in the instruction register.

Processors

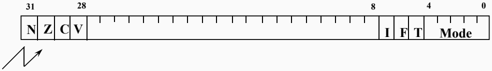
Computer Architecture



State Machine



Status Register



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

- **Condition Code Flags**

  N = **N**egative result from ALU flag.
  Z = **Z**ero result from ALU flag.
  C = ALU operation **C**arried out
  V = ALU operation o**V**erflowed

- **Mode Bits**

  M[4:0] define the processor mode.

- **Interrupt Disable bits.**

  I = 1, disables the IRQ.
  F = 1, disables the FIQ.

- **T Bit (Architecture v4T only)**

  T = 0, Processor in ARM state
  T = 1, Processor in Thumb state

Flags

	Logical Instruction	Arithmetic Instruction
Flag		
Negative (N='1')	No meaning	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result is all zeroes	Result of operation was zero
Carry (C='1')	After Shift operation '1' was left in carry flag	Result was greater than 32 bits
oVerflow (V='1')	No meaning	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

Conditions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
Condition				0	0	I	OPCODE				S	Rn				Rs				OPERAND-2												Data processing

0000 = EQ - Z set (equal)

0001 = NE - Z clear (not equal)

0010 = HS / CS - C set (unsigned higher or same)

0011 = LO / CC - C clear (unsigned lower)

0100 = MI -N set (negative)

0101 = PL - N clear (positive or zero)

0110 = VS - V set (overflow)

0111 = VC - V clear (no overflow)

1000 = HI - C set and Z clear (unsigned higher)

1001 = LS - C clear or Z (set unsigned lower or same)

1010 = GE - N set and V set, or N clear and V clear (>or =)

1011 = LT - N set and V clear, or N clear and V set (>)

1100 = GT - Z clear, and either N set and V set, or N clear and V set (>)

1101 = LE - Z set, or N set and V clear,or N clear and V set (<, or =)

1110 = AL - always

1111 = NV - reserved.

Instructions

- `OP-Code`
  - Decides operation
- `Operand`
  - OP-Code dependent
  - "Parameters"
- Each instruction on ARM `32 bits`

Instruction ARM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
Condition				0	0	I	OPCODE				S	Rn				Rs				OPERAND-2												Data processing
Condition				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply
Condition				0	0	0	0	1	U	A	S	Rd HIGH				Rd LOW				Rs				1	0	0	1	Rm				Long Multiply
Condition				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				Swap
Condition				0	1	I	P	U	B	W	L	Rn				Rd				OFFSET												Load/Store - Byte/Word
Condition				1	0	0	P	U	B	W	L	Rn				REGISTER LIST																Load/Store Multiple
Condition				0	0	0	P	U	1	W	L	Rn				Rd				OFFSET 1				1	S	H	1	OFFSET 2				Halfword Transfer Imm Off
Condition				0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				Halfword Transfer Reg Off
Condition				1	0	1	L	BRANCH OFFSET																								Branch
Condition				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				Branch Exchange
Condition				1	1	0	P	U	N	W	L	Rn				CRd				CPNum				OFFSET								COPROCESSOR DATA XFER
Condition				1	1	1	0	Op-1				CRn				CRd				CPNum				OP-2			0	CRm				COPROCESSOR DATA OP
Condition								OP-1			L	CRn				Rd				CPNum				OP-2			1	CRm				COPROCESSOR REG XFER
Condition				1	1	1	1	SWI NUMBER																								Software Interrupt

Reference Manual

Instructions

- Data Transfer
  - `LDR`, `STR`
- Flow Control
  - `B`, `CMP`
- Arithmetic
  - `ADD`, `MUL`

LDR/STR

4.9.8 Assembler syntax

```
<LDR|STR>{cond}{B}{T} Rd,<Address>
```

where:

LDR	load from memory into a register
STR	store from a register into memory
{cond}	two-character condition mnemonic. See ***Table 4-2: Condition code summary*** on page 4-5.
{B}	if B is present then byte transfer, otherwise word transfer
{T}	if T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.
Rd	is an expression evaluating to a valid register number.
Rn and Rm	are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for pipelining. In this case base write-back should not be specified.
<Address>	can be:

1. An expression which generates an address:

   ```
   <expression>
   ```

   The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2. A pre-indexed addressing specification:

`[Rn]`	offset of zero
`[Rn,<#expression>]{!}`	offset of <expression> bytes
`[Rn,{+/-}Rm{,<shift>}]{!}`	offset of +/- contents of index register, shifted by <shift>

3. A post-indexed addressing specification:

`[Rn],<#expression>`	offset of <expression> bytes
`[Rn],{+/-}Rm{,<shift>}`	offset of +/- contents of index register, shifted as by <shift>.

<shift> general shift operation (see data processing instructions) but you cannot specify the shift amount by a register.

Addressing Modes

`ldr r0,[r1,#4]` Load word addressed by R1+4.

`str r0,[r1],#4` Store R0 to word addressed by R1. Increment R1 by 4.

`ldr r0,[r1,#4]!` Load word addressed by R1+4. Increment R1 by 4.

`ldr r0,=label` Load address of label `label` into R0

ADD

Code

```
_start:
        add     r0, r1
        add     r0, #4
        add     r0, [r1, #3]
```

MUL

4.7 Multiply and Multiply-Accumulate (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in ***Table 4-2: Condition code summary*** on page 4-5. The instruction encoding is shown in ***Figure 4-12: Multiply instructions***.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

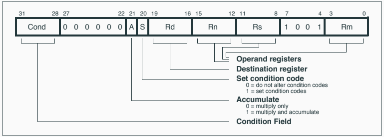


Figure 4-12: Multiply instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.

The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances.

Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits - the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

Operand AOperand BResult

0xFFFFFFF6 0x00000010xFFFFFF38

If the operands are interpreted as signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFF38

If the operands are interpreted as unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFF38.

4.7.1 Operand restrictions

The destination register Rd must not be the same as the operand register Rm. R15 must not be used as an operand or as the destination register.

Branch

4.4 Branch and Branch with Link (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined ***Table 4-2: Condition code summary*** on page 4-5. The instruction encoding is shown in ***Figure 4-3: Branch instructions***, below.

Figure 4-3: Branch instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

4.4.1 The link bit

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

4.4.2 Instruction cycle times

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are as defined in ***6.2 Cycle Types*** on page 6-3.

Code

Code

```
        .section        .text
        .align              2
        .global              _start
_start:
        ldr         r0, =matrix

        ldr         r1, [r0]            @ a
        ldr         r2, [r0, #12]           @ d
        mul         r7, r2,r1           @ a*d

        ldr         r1, [r0, #4]           @ b
        ldr         r2, [r0, #8]           @ c
        mul         r6, r2, r1           @ b*c
```

Summary

Summary

- This was just a very quick intro to CPUs
- You learn this by doing
- Lab on Monday we will play with these things