CISC260 Machine Organization and Assembly Language

ARM and Its ISA

[quiz] For any logic devise, its output is solely determined by the input. Correct or not?

A. Yes
B. No
C. Depends
D. Don't know

Logic devised can be of 2 types: one is combinational and the other is sequential. While for combinational, the output is determined by the input solely, for sequential, the output is determined by the order/history and the input

ARM CPU Overview

PC: Program Counter

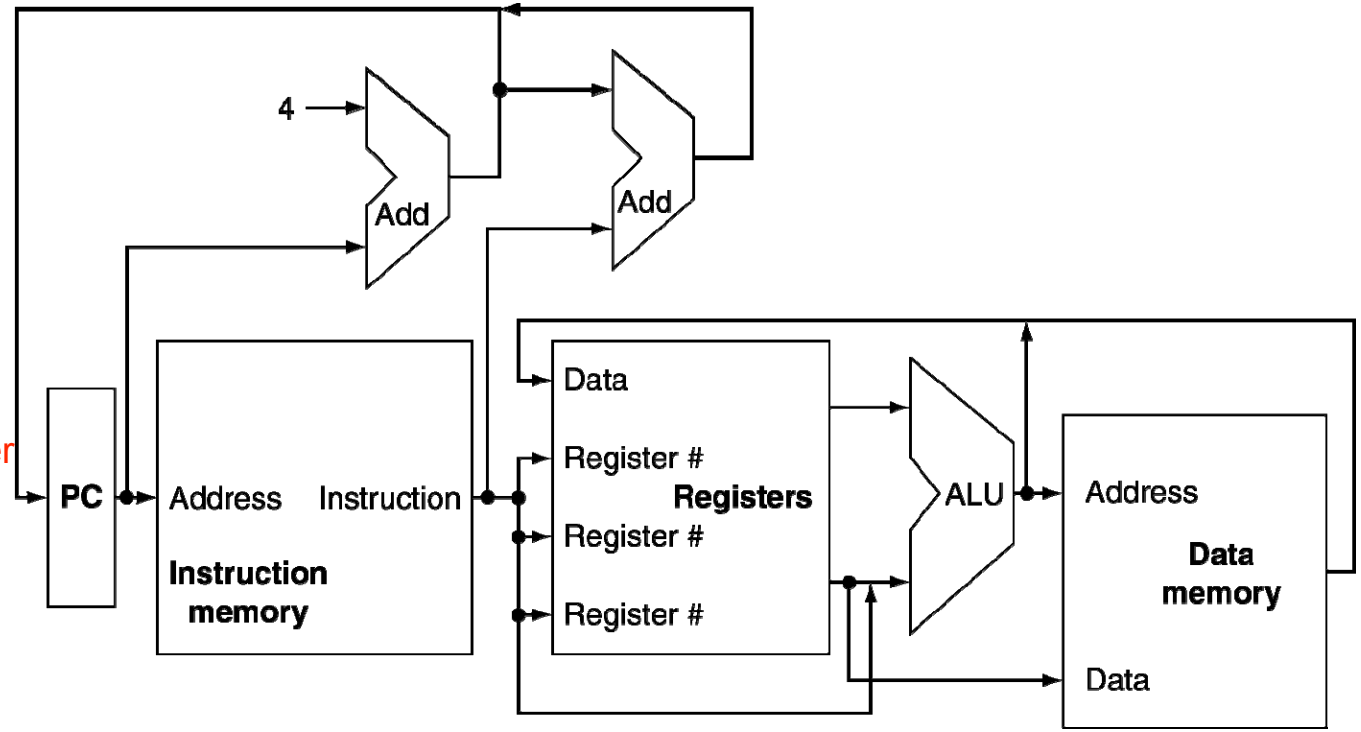
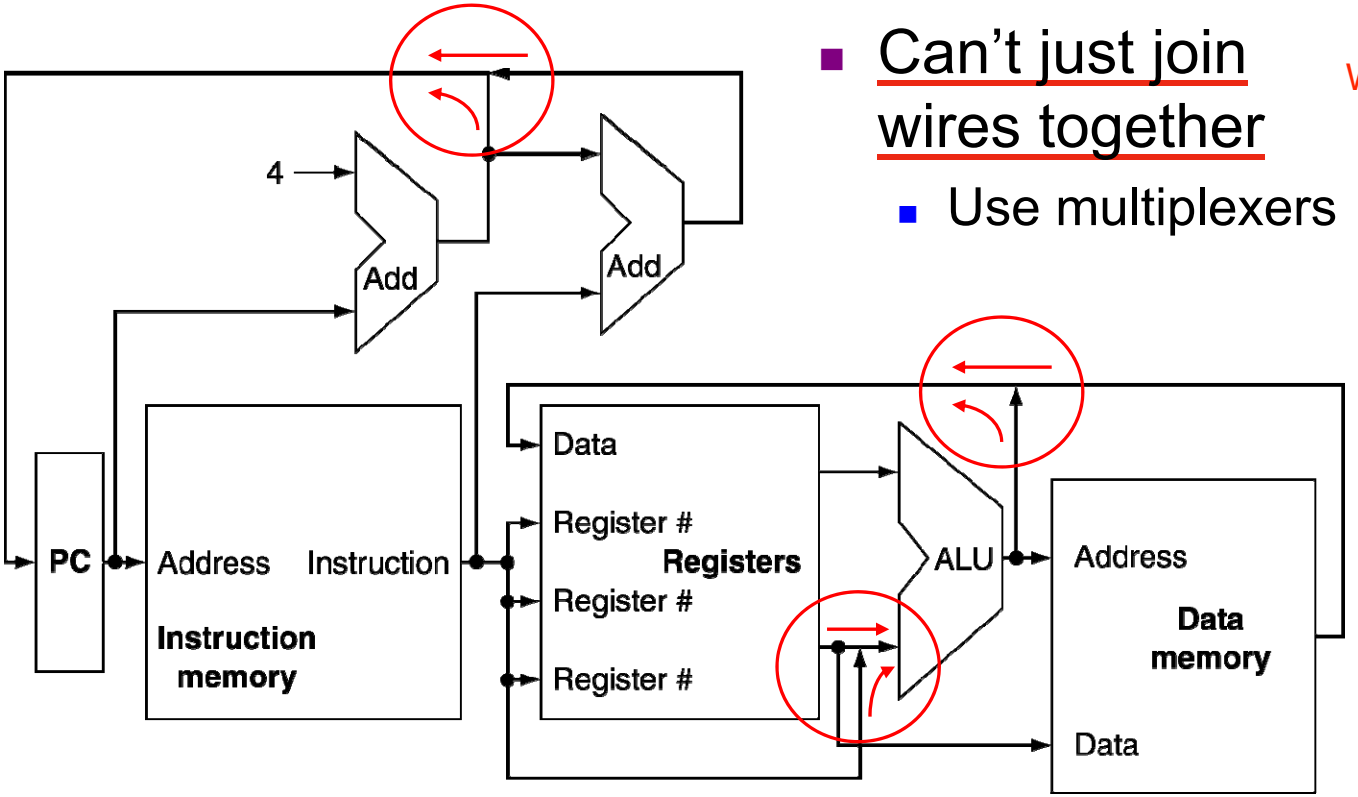


Figure 4.1

Note: separation of instruction memory and data memory is for ease of conceptual understanding, so is the separation of PC from the register file.

Multiplexers



- Can't just join wires together
  - Use multiplexers

WHY

Control & Data Path

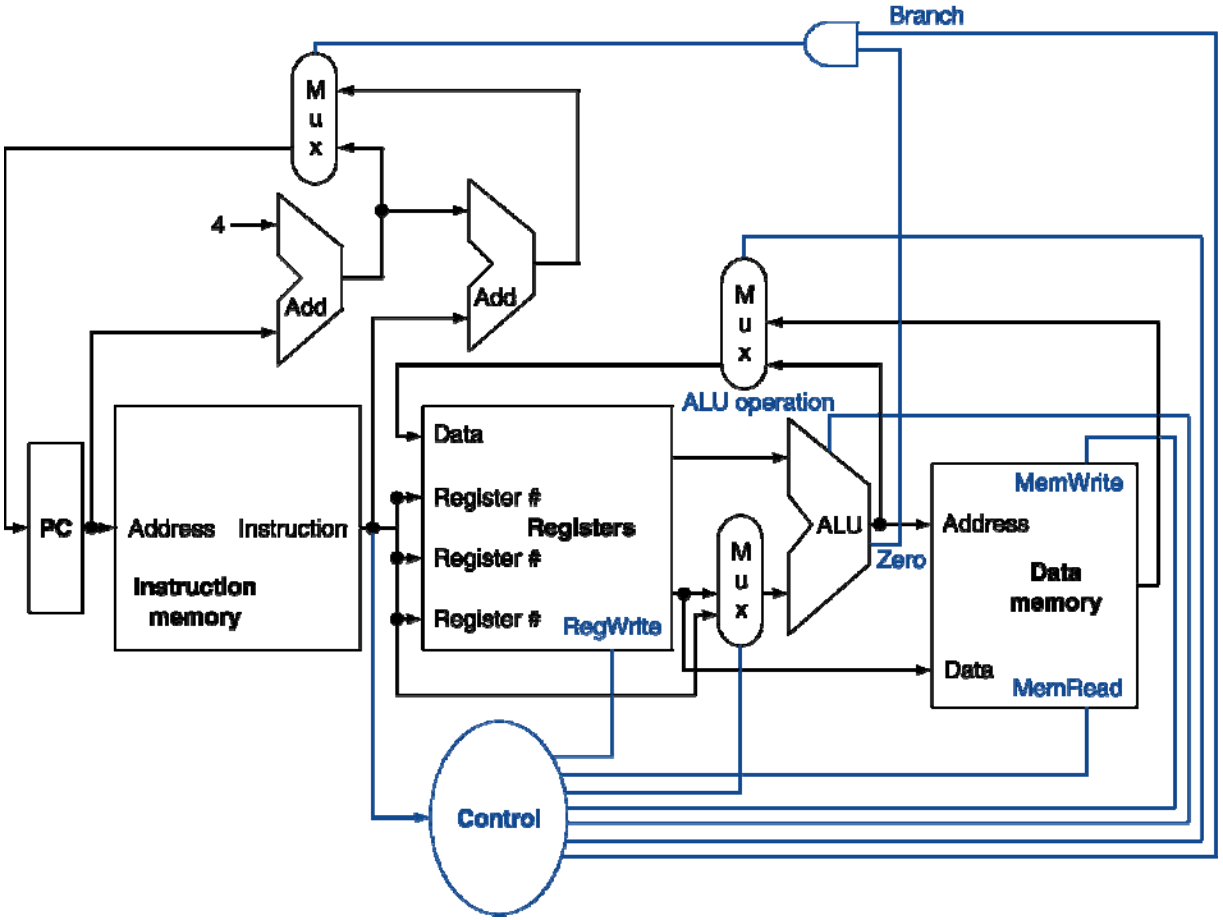


Figure 4.2

Note: The following is for the 32-bit ARM7, see Chapter 02_COD 4e ARM

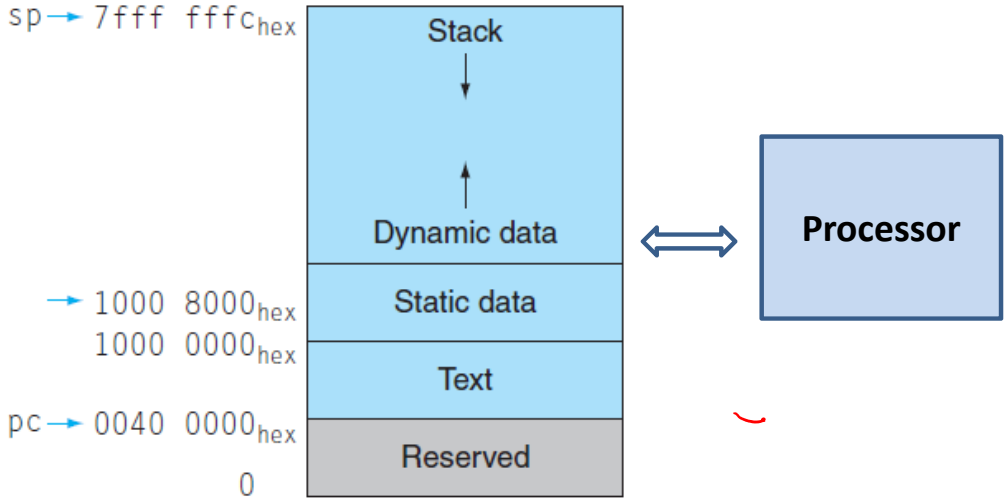


FIGURE 2.13 Typical ARM memory allocation for program and data. These addresses are only a software convention, and not part of the ARM architecture. The stack pointer is initialized to $7fff\ fffc_{hex}$ and grows down toward the data segment. At the other end, the program code ("text") starts at $0040\ 0000_{hex}$. The static data starts at $1000\ 0000_{hex}$. Dynamic data, allocated by `malloc` in C and by `new` in Java, is next. It grows up toward the stack in an area called the heap.

what does "halfword" mean
machine dependent

ARM operands

Name	Example	Comments
16 registers	r0, r1, r2,..., r11, r12, sp, lr, pc	Fast locations for data. In ARM, data must be in registers to perform arithmetic, register
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. ARM uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

ARM assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	ADD r1,r2,r3	r1 = r2 – r3	3 register operands
	subtract	SUB r1,r2,r3	r1 = r2 + r3	3 register operands
Data transfer	load register	LDR r1, [r2,#20]	r1 = Memory[r2 + 20]	Word from memory to register
	store register	STR r1, [r2,#20]	Memory[r2 + 20] = r1	Word from memory to register
	load register halfword	LDRH r1, [r2,#20]	r1 = Memory[r2 + 20]	Halfword memory to register
	load register halfword signed	LDRHS r1, [r2,#20]	r1 = Memory[r2 + 20]	Halfword memory to register
	store register halfword	STRH r1, [r2,#20]	Memory[r2 + 20] = r1	Halfword register to memory
	load register byte	LDRB r1, [r2,#20]	r1 = Memory[r2 + 20]	Byte from memory to register
	load register byte signed	LDRBS r1, [r2,#20]	r1 = Memory[r2 + 20]	Byte from memory to register
	store register byte	STRB r1, [r2,#20]	Memory[r2 + 20] = r1	Byte from register to memory
	swap	SWP r1, [r2,#20]	r1 = Memory[r2 + 20], Memory[r2 + 20] = r1	Atomic swap register and memory
	mov	MOV r1, r2	r1 = r2	Copy value into register
Logical	and	AND r1, r2, r3	r1 = r2 & r3	Three reg. operands; bit-by-bit AND
	or	ORR r1, r2, r3	r1 = r2 | r3	Three reg. operands; bit-by-bit OR
	not	MVN r1, r2	r1 = ~ r2	Two reg. operands; bit-by-bit NOT
	logical shift left (optional operation)	LSL r1, r2, #10	r1 = r2 << 10	Shift left by constant
	logical shift right (optional operation)	LSR r1, r2, #10	r1 = r2 >> 10	Shift right by constant
Conditional Branch	compare	CMP r1, r2	cond. flag = r1 – r2	Compare for conditional branch
	branch on EQ, NE, LT, LE, GT, GE, LO, LS, HI, HS, VS, VC, MI, PL	BEQ 25	if (r1 == r2) go to PC + 8 + 100	Conditional Test; PC-relative
Unconditional Branch	branch (always)	B 2500	go to PC + 8 + 10000	Branch
	branch and link	BL 2500	r14 = PC + 4; go to PC + 8 + 10000	For procedure call

FIGURE 2.1 ARM assembly language revealed in this chapter. This information is also found in Column 1 of the ARM Reference Data Card at the front of this book.

Features of ARM instruction set

- Load-store architecture
- 3-address instructions
- Conditional execution of every instruction
- Possible to load/store multiple register at once
- Possible to combine shift and ALU operations in a single instruction

ARM Instruction Fields

Cond	F	I	Opcode	S	Rn	Rd	Operand2
4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits

Here is the meaning of each name of the fields in ARM instructions:

- *Opcode:* Basic operation of the instruction, traditionally called the opcode.
- *Rd:* The register destination operand. It gets the result of the operation.
- *Rn:* The first register source operand.
- *Operand2:* The second source operand.
- *I:* Immediate. If I is 0, the second source operand is a register. If I is 1, the second source operand is a 12-bit immediate. (Section 2.10 goes into details on ARM immediates, but for now we'll just assume its just a plain constant.)
- *S:* Set Condition Code. Described in Section 2.7, this field is related to conditional branch instructions.
- *Cond:* Condition. Described in Section 2.7, this field is related to conditional branch instructions.
- *F:* Instruction Format. This field allows ARM to different instruction formats when needed.

Let's look at the add word instruction from page 83 that has a constant operand:

```
ADD     r3,r3,#4        ; r3 = r3 + 4
```

As you might expect, the constant 4 is placed in the Operand2 field and the I field is set to 1. We make a small change the binary version from before.

14	0	1	4	0	3	3	4

```
LDR     r5,[r3, #32]    ; Temporary reg r5 gets A[8]
```

Loads and stores use a different instruction format from above, with just 6 fields:

Cond	F	Opcode	Rn	Rd	Offset12
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

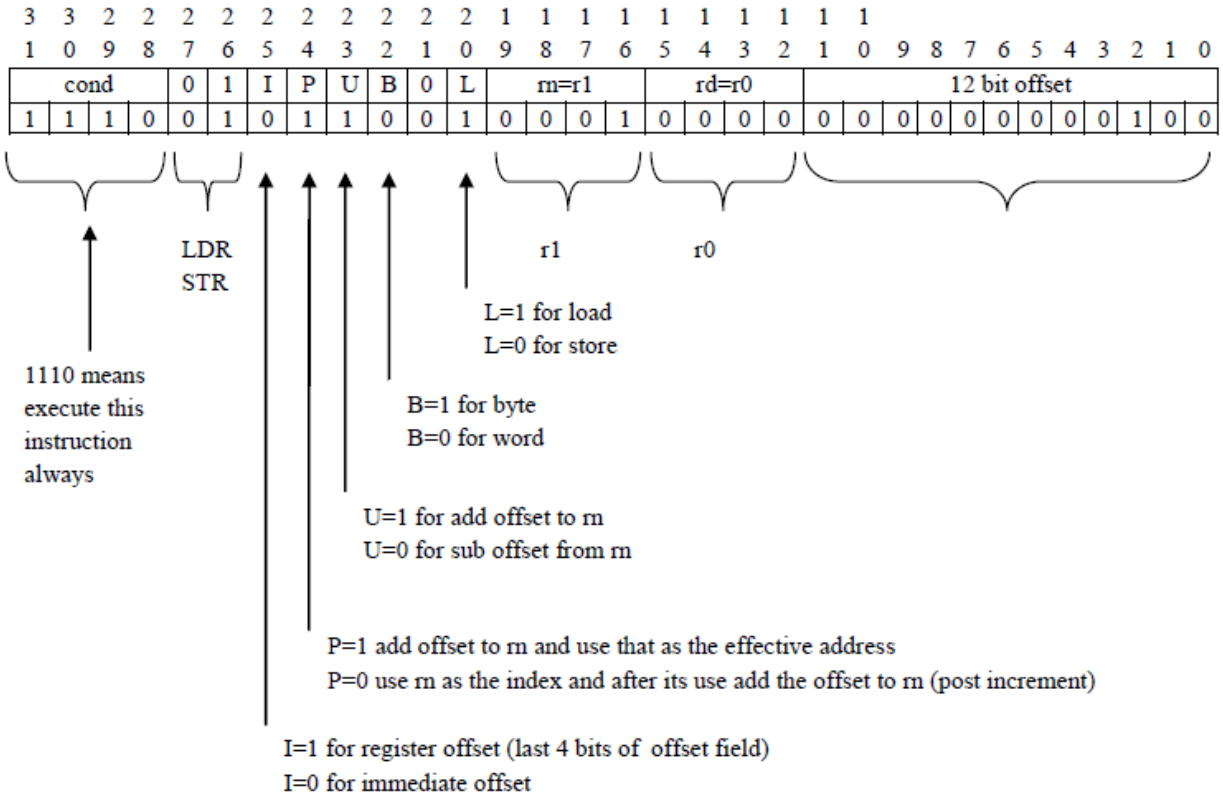
To tell ARM that the format is different, the F field now as 1, meaning that this is a data transfer instruction format. The opcode field has 24, showing that this instruction does load word. The rest of the fields are straightword: Rn field has 3 for the base register, the Offset12 field has 32 as the offset to add to the base register, and the Rd field has 5 for the *destination* register, which receives the result of the load:

14	1	24	3	5	32
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

ldr r0, [r1, #4] ; r0 = the word of data pointed to by r1 + 4

The hex value of the machine instruction is: **E5 91 00 04**

This is the binary value of that machine instruction. We describe the fields left to right.



Instruction	Format	Cond	F	I	op	S	Rn	Rd	Operand2
ADD	DP	14	0	0	4_{ten}	0	reg	reg	reg
SUB (subtract)	DP	14	0	0	$2s_{ten}$	0	reg	reg	reg
ADD (immediate)	DP	14	0	1	4_{ten}	0	reg	reg	constant
LDR (load word)	DT	14	1	n.a.	24_{ten}	n.a.	reg	reg	address
STR (store word)	DT	14	1	n.a.	25_{ten}	n.a.	reg	reg	address

FIGURE 2.5 ARM instruction encoding. In the table above, "reg" means a register number between 0 and 15, "constant" means a 12-bit constant, "address" means a 12-bit address. "n.a." (not applicable) means this field does not appear in this format, and Op stands for opcode.

ARM machine language

Name	Format	Example								Comments
ADD	DP	14	0	0	4	0	2	1	3	ADD r1,r2,r3
SUB	DP	14	0	0	2	0	2	1	3	SUB r1,r2,r3
LDR	DT	14	1	24			2	1	100	LDR r1,100(r2)
STR	DT	14	1	25			2	1	100	STR r1,100(r2)
Field size		4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits	All ARM instructions are 32 bits long
DP format	DP	Cond	F	I	Opcode	S	Rn	Rd	Operand2	Arithmetic instruction format
DT format	DT	Cond	F	Opcode			Rn	Rd	Offset12	Data transfer format

FIGURE 2.6 ARM architecture revealed through Section 2.5. The two ARM instruction formats so far are DP and DT. The last 16 bits have the same sized fields: both contain an *Rn* field, giving one of the sources; and an *Rd* field, specifying the destination register.

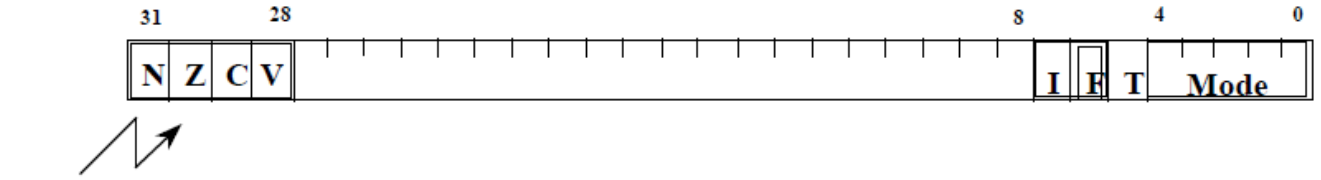
Format of Branching Instructions

Name	Format	Example								Comments
Field size		4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits	All ARM instructions are 32 bits long
DP format	DP	Cond	F	I	Opcode	S	Rn	Rd	Operand2	Arithmetic instruction format
DT format	DT	Cond	F	Opcode			Rn	Rd	Offset12	Data transfer format
Field size		4 bits	2 bits	2 bits	24 bits					
BR format	BR	Cond	F	Opcode	signed_immed_24					B and BL instructions

Value	Meaning	Value	Meaning
0	EQ (EQual)	8	HI (unsigned HIgher)
1	NE (Not Equal)	9	LS (unsigned Lower or Same)
2	HS (unsigned Higher or Same)	10	GE (signed Greater than or Equal)
3	LO (unsigned LOwer)	11	LT (signed Less Than)
4	MI (MInus, <0)	12	GT (signed Greater Than)
5	PL - (PLus, >=0)	13	LE (signed Less Than or Equal)
6	VS (oVerflow Set, overflow)	14	AL (Always)
7	VC (oVerflow Clear, no overflow)	15	NV (reserved)

FIGURE 2.9.5 Encodings of Options for Cond field.

The Program Status Registers (CPSR and SPSRs)



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.

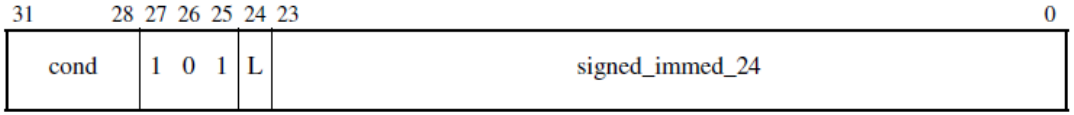
* **Interrupt Disable bits.**

  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

B, BL



B (Branch) and BL (Branch and Link) cause a branch to a target address, and provide both conditional and unconditional changes to program flow.

BL also stores a return address in the link register, R14 (also known as LR).

Syntax

B{L}{<cond>} <target_address>

where:

L — Causes the L bit (bit 24) in the instruction to be set to 1. The resulting instruction stores a return address in the link register (R14). If L is omitted, the L bit is 0 and the instruction simply branches without storing a return address.

<cond> — Is the condition under which the instruction is executed. The conditions are defined in *The condition field* on page A3-3. If <cond> is omitted, the AL (always) condition is used.

<target_address>

Specifies the address to branch to. The branch target address is calculated by:

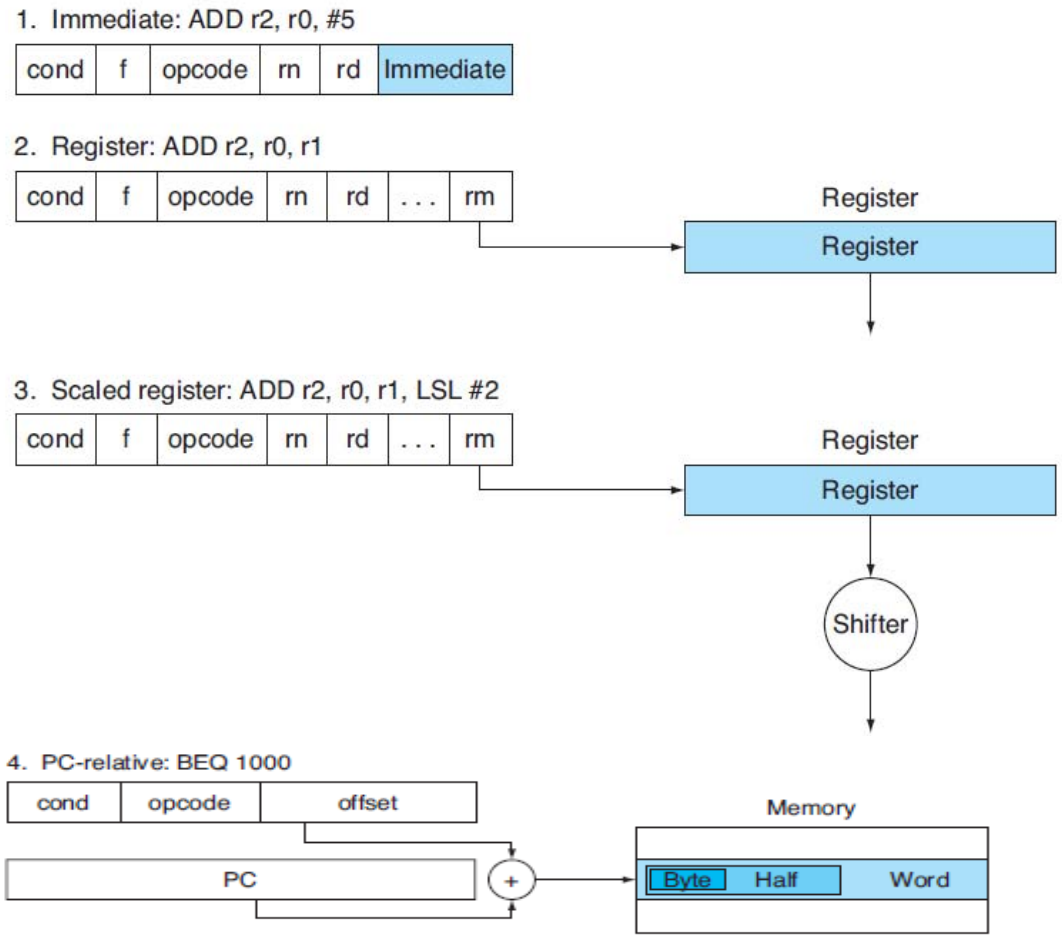
1. Sign-extending the 24-bit signed (two's complement) immediate to 30 bits.
2. Shifting the result left two bits to form a 32-bit value.
3. Adding this to the contents of the PC, which contains the address of the branch instruction plus 8 bytes.

The instruction can therefore specify a branch of approximately ±32MB (see *Usage* on page A4-11 for precise range).

Name	Register number	Usage	Preserved on call?
a1-a2	0–1	Argument / return result / scratch register	no
a3-a4	2–3	Argument / scratch register	no
v1-v8	4–11	Variables for local routine	yes
ip	12	Intra-procedure-call scratch register	no
sp	13	Stack pointer	yes
lr	14	Link Register (Return address)	yes
pc	15	Program Counter	n.a.

FIGURE 2.14 ARM register conventions.

ARM Addressing Modes



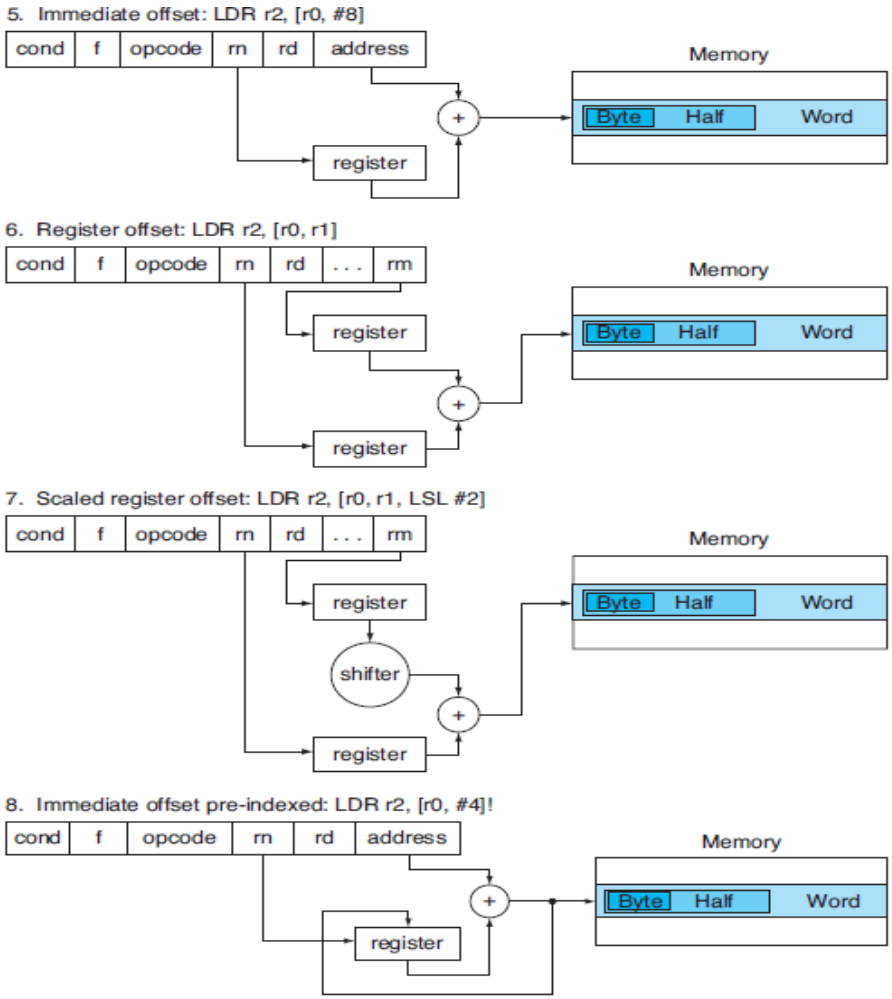


FIGURE 2.17 Illustration of the twelve ARM addressing modes. (*continued*)

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

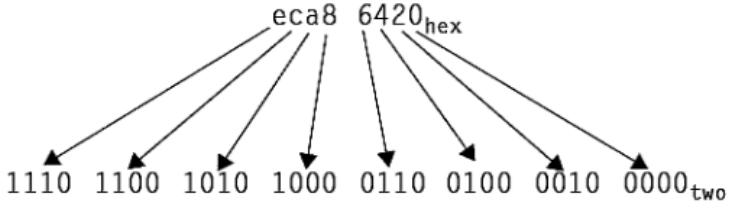
Binary-to-Hexadecimal and Back

Convert the following hexadecimal and binary numbers into the other base:

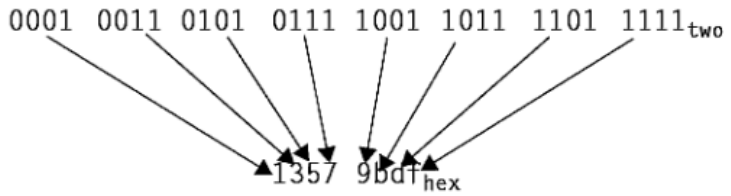
$\texttt{eca8 6420}_{hex}$

$0001\ 0011\ 0101\ 0111\ 1001\ 1011\ 1101\ 1111_{two}$

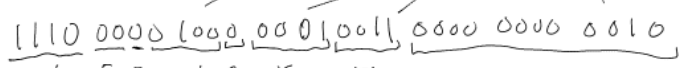
Just a table lookup one way:



And then the other direction too:



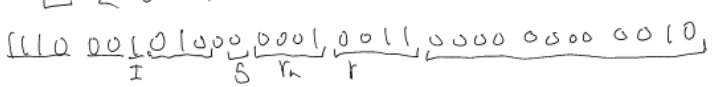
E0813002 : add r3, r1, r2



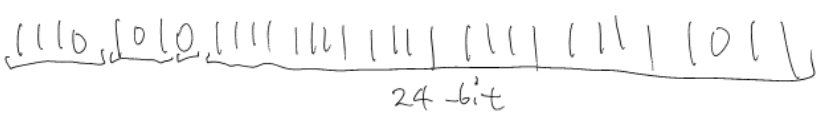
C0813002 : addgt r3, r1, r2

1100

E2813002 : add r3, r1, #2



EAFFFFFB : b -12

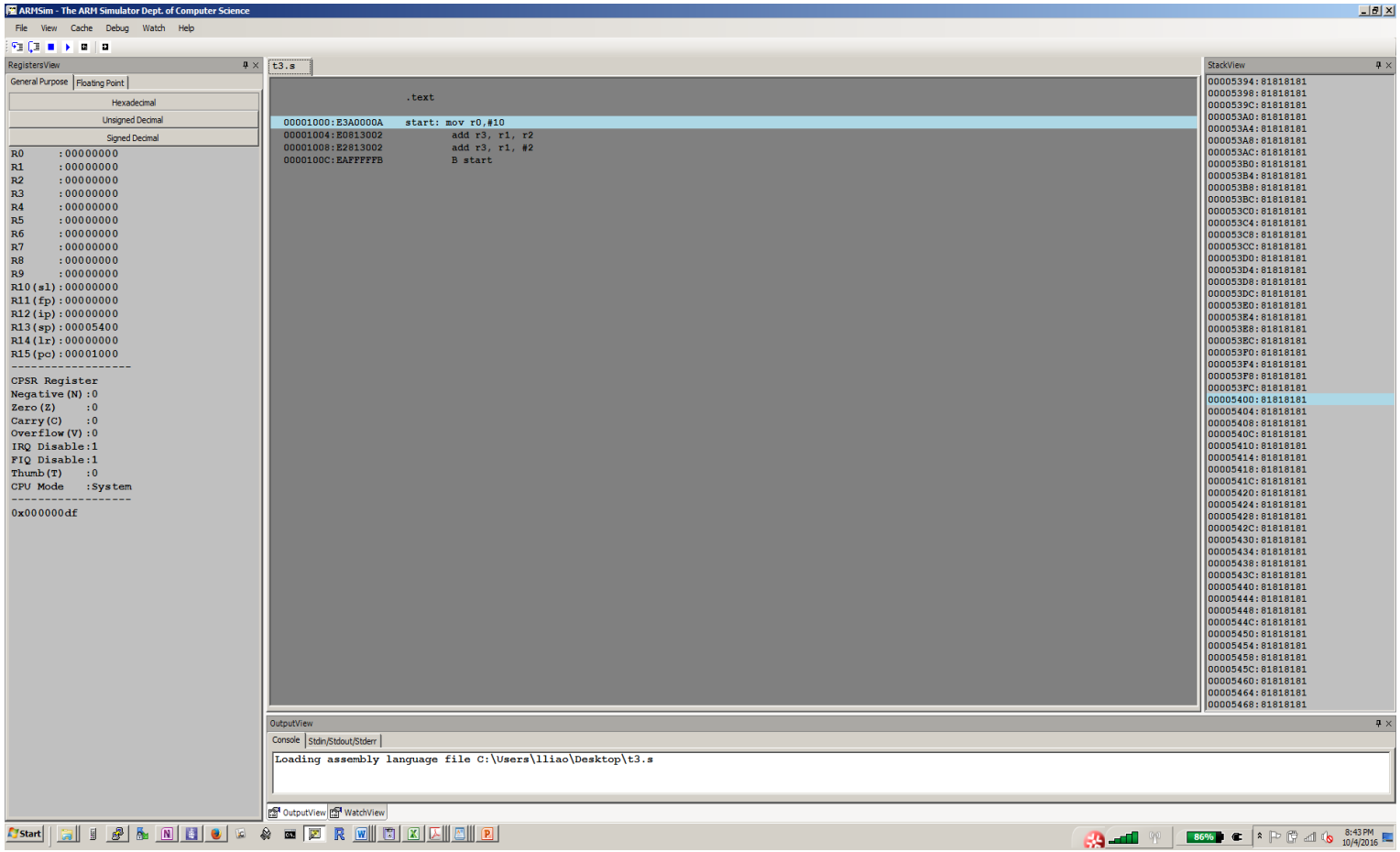


a. Sign-ext 30 1011 00 = -20

1111 1111 1111 1111 1111 1111 1111 0100 = -20

+) old PC + 8

PC = old PC - 12


ARMSim - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help
RegistersView
General Purpose Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00000000
R1 :00000000
R2 :00000000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00001000
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x000000df
t3.s
.text
00001000:E3A0000A start: mov r0,#10
00001004:E0813002 add r3, r1, r2
00001008:E2813002 add r3, r1, #2
0000100C:EAFFFFFB B start
StackView
00005394:81818181
00005398:81818181
0000539C:81818181
000053A0:81818181
000053A4:81818181
000053A8:81818181
000053AC:81818181
000053B0:81818181
000053B4:81818181
000053B8:81818181
000053BC:81818181
000053C0:81818181
000053C4:81818181
000053C8:81818181
000053CC:81818181
000053D0:81818181
000053D4:81818181
000053D8:81818181
000053DC:81818181
000053E0:81818181
000053E4:81818181
000053E8:81818181
000053EC:81818181
000053F0:81818181
000053F4:81818181
000053F8:81818181
000053FC:81818181
00005400:81818181
00005404:81818181
00005408:81818181
0000540C:81818181
00005410:81818181
00005414:81818181
00005418:81818181
0000541C:81818181
00005420:81818181
00005424:81818181
00005428:81818181
0000542C:81818181
00005430:81818181
00005434:81818181
00005438:81818181
0000543C:81818181
00005440:81818181
00005444:81818181
00005448:81818181
0000544C:81818181
00005450:81818181
00005454:81818181
00005458:81818181
0000545C:81818181
00005460:81818181
00005464:81818181
00005468:81818181
OutputView
Console Stdin/Stdout/Stderr
Loading assembly language file C:\Users\lliao\Desktop\t3.s
OutputView WatchView
Start
86%
8:43 PM
10/4/2016

Refer to the *ARM Architectural Reference Manual* for more information about the ARM instruction set formats.

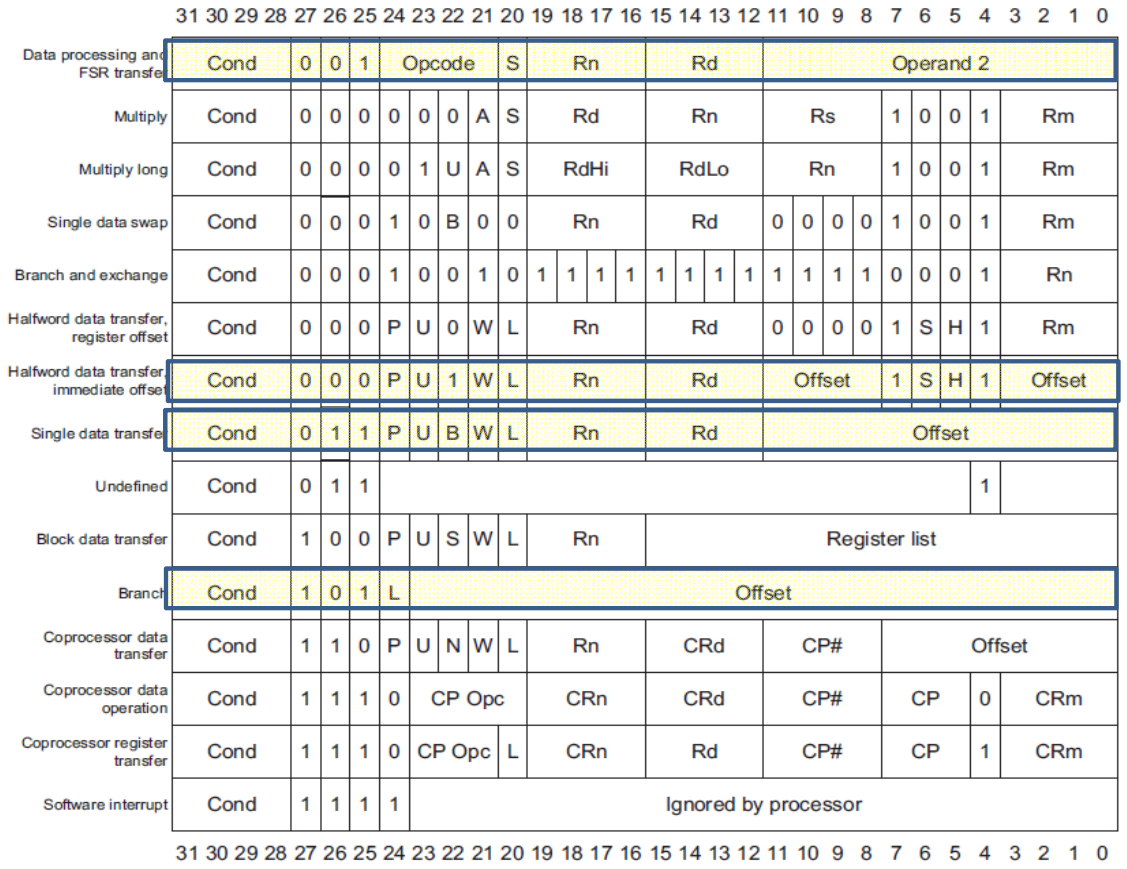
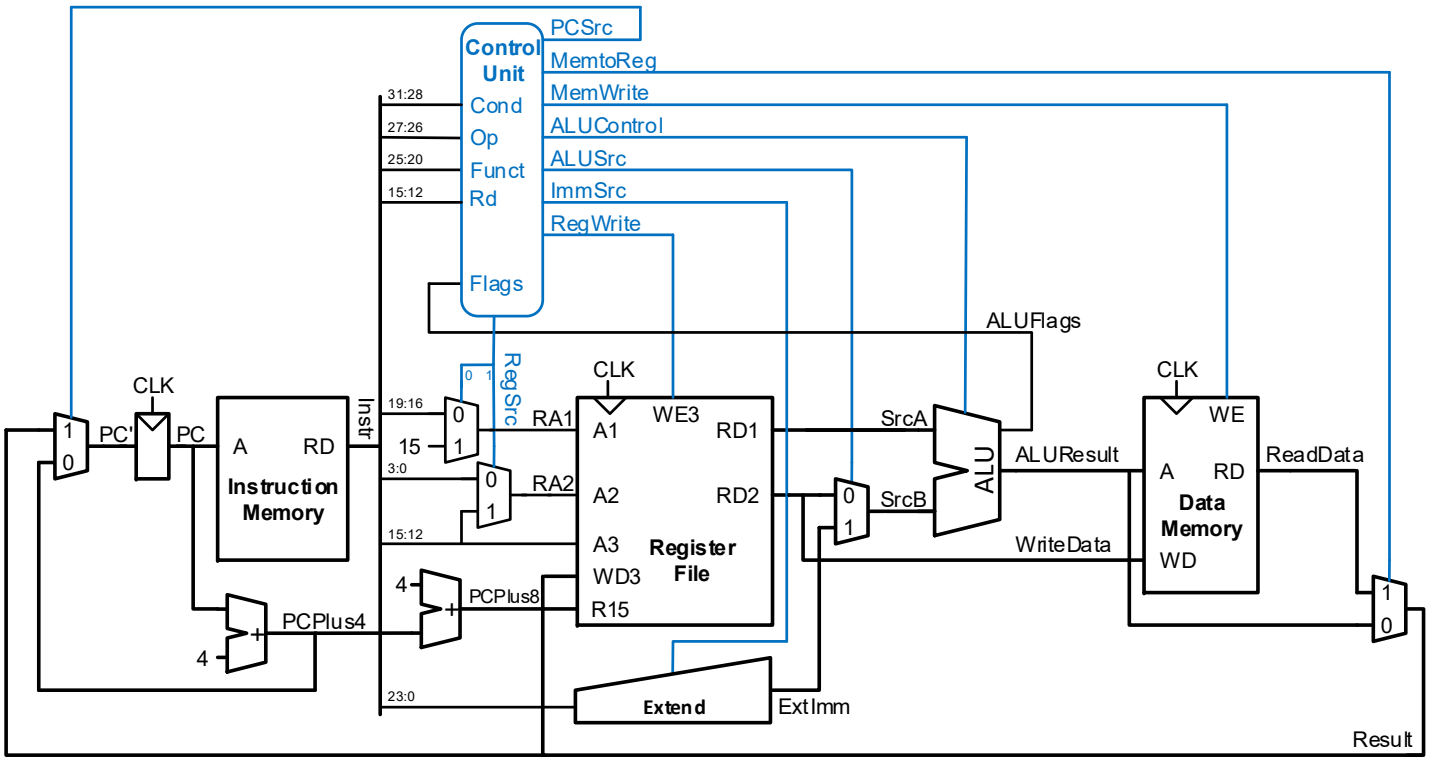


Figure 1-5 ARM instruction set formats


Control Unit
PCSrc
MemtoReg
MemWrite
ALUControl
ALUSrc
ImmSrc
RegWrite
Cond
Op
Funct
Rd
Flags
31:28
27:26
25:20
15:12
ALUFlags
CLK
PC'
PC
A
RD
Instruction Memory
Instr
19:16
15
3:0
15:12
RegSrc
RA1
RA2
A1
A2
A3
WD3
R15
WE3
RD1
RD2
Register File
CLK
SrcA
SrcB
ALU
ALUResult
WriteData
CLK
WE
A
RD
Data Memory
WD
ReadData
4
PCPlus4
PCPlus8
Extend
23:0
ExtImm
Result