Hand Assembly

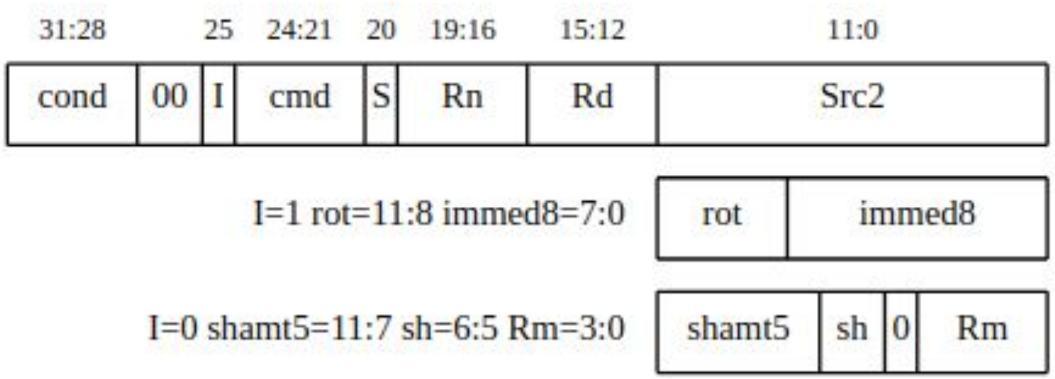
Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

Hand Assembly

- An assembler translates assembly language into machine code (i.e., ones and zeros)
- All ARMv4 instructions are encoded in 32 bits
  - ADD R0, R0, R1 -> 0010....0001

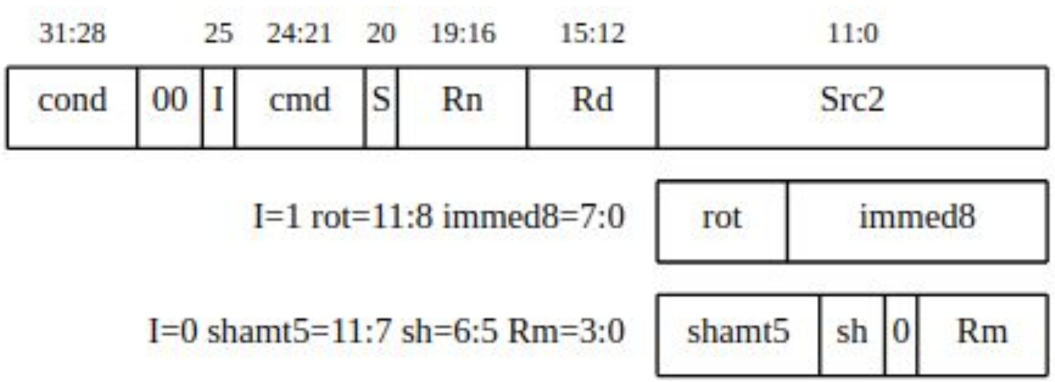
Hand Assembly

- The data processing instruction (bits 27:26 = 00)
    - Memory = 01
    - Branch = 10
    - Reserved = 11
- Data processing instruction use the following format:



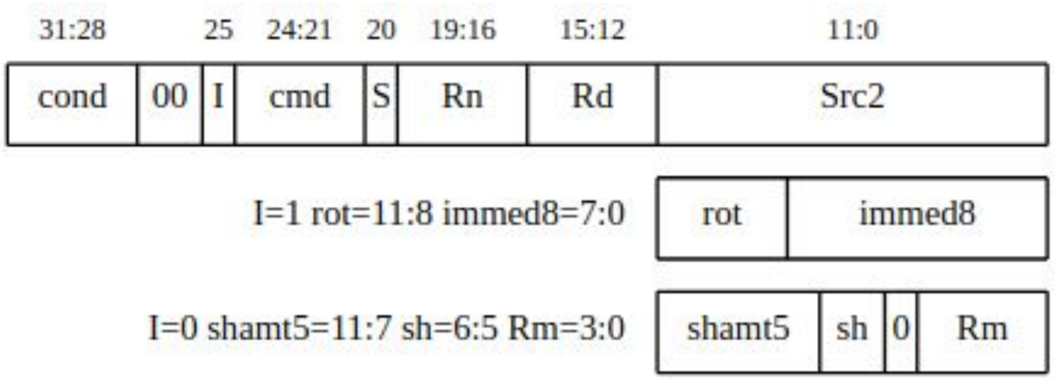
Hand Assembly

- The cond field (31:28) determines if the instruction is executed
- If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored



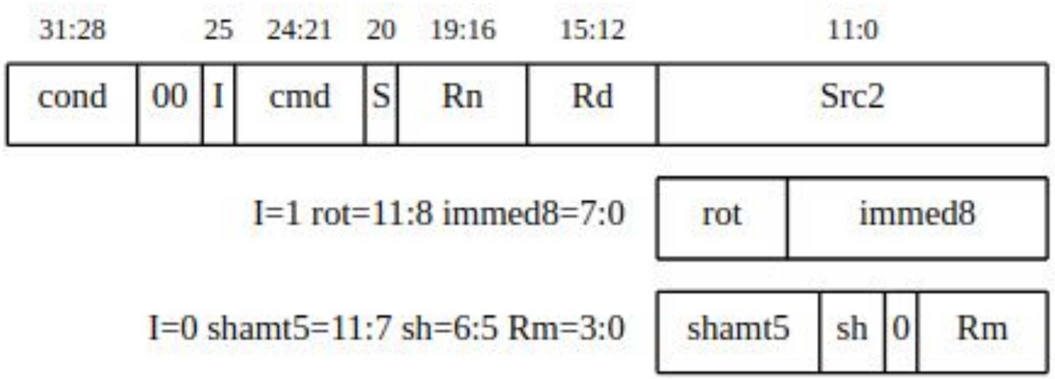
Hand Assembly

- The cmd field (24:21) determines the type of data processing instruction
- The I field (25) determine if the second operand is a register or an immediate value
- If a I=0 then a register operand is selected

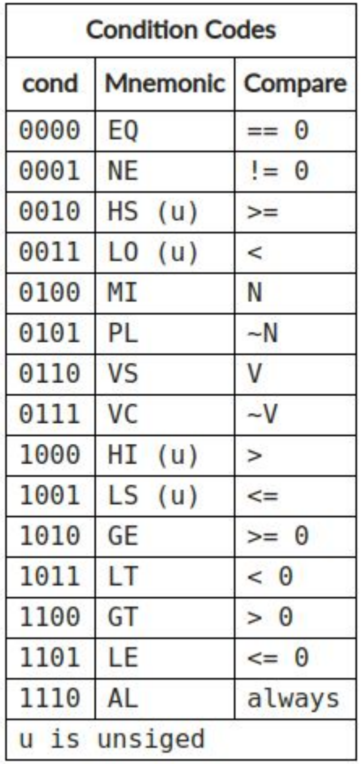


Hand Assembly

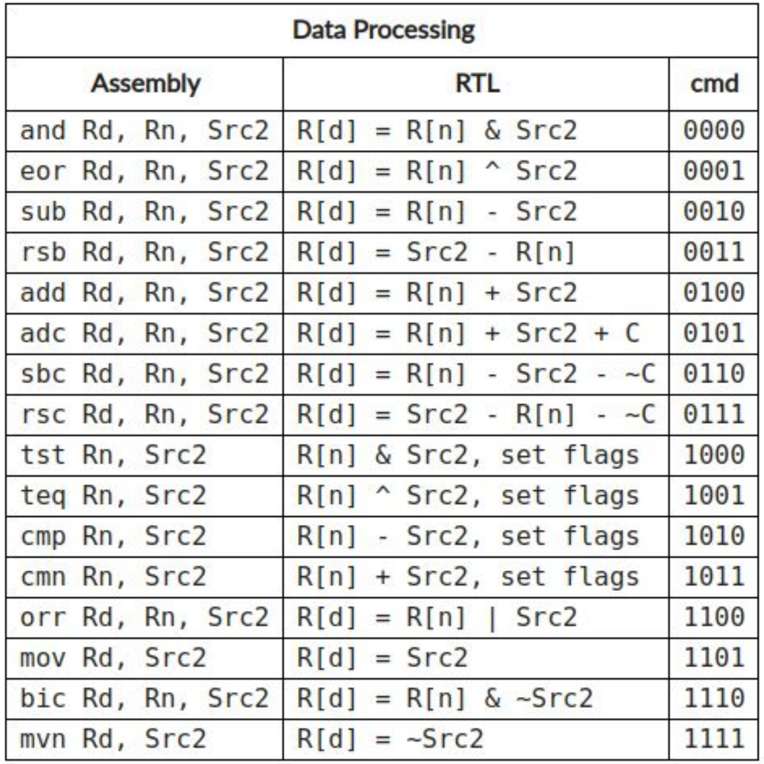
- S can also be added to each instruction, to determine if the condition flags are updated



Hand Assembly



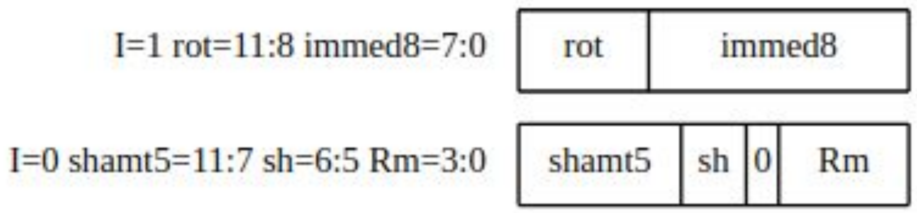
Condition Codes		
cond	Mnemonic	Compare
0000	EQ	== 0
0001	NE	!= 0
0010	HS (u)	>=
0011	LO (u)	<
0100	MI	N
0101	PL	~N
0110	VS	V
0111	VC	~V
1000	HI (u)	>
1001	LS (u)	<=
1010	GE	>= 0
1011	LT	< 0
1100	GT	> 0
1101	LE	<= 0
1110	AL	always
u is unsiged		



Data Processing		
Assembly	RTL	cmd
and Rd, Rn, Src2	R[d] = R[n] & Src2	0000
eor Rd, Rn, Src2	R[d] = R[n] ^ Src2	0001
sub Rd, Rn, Src2	R[d] = R[n] - Src2	0010
rsb Rd, Rn, Src2	R[d] = Src2 - R[n]	0011
add Rd, Rn, Src2	R[d] = R[n] + Src2	0100
adc Rd, Rn, Src2	R[d] = R[n] + Src2 + C	0101
sbc Rd, Rn, Src2	R[d] = R[n] - Src2 - ~C	0110
rsc Rd, Rn, Src2	R[d] = Src2 - R[n] - ~C	0111
tst Rn, Src2	R[n] & Src2, set flags	1000
teq Rn, Src2	R[n] ^ Src2, set flags	1001
cmp Rn, Src2	R[n] - Src2, set flags	1010
cmn Rn, Src2	R[n] + Src2, set flags	1011
orr Rd, Rn, Src2	R[d] = R[n] | Src2	1100
mov Rd, Src2	R[d] = Src2	1101
bic Rd, Rn, Src2	R[d] = R[n] & ~Src2	1110
mvn Rd, Src2	R[d] = ~Src2	1111

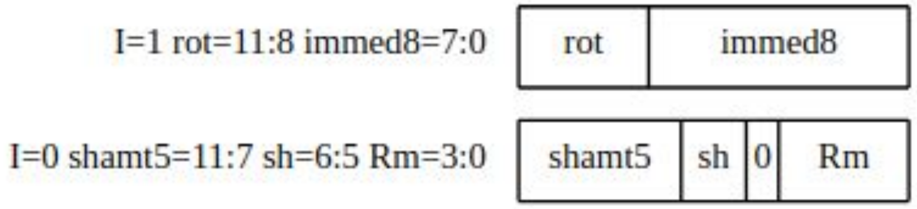
Hand Assembly

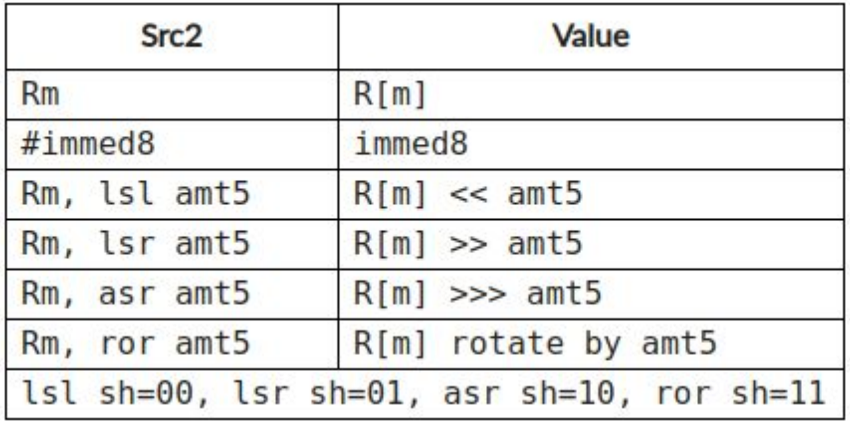
- In assembly language, Src2 has the following forms:
- I = 1 -> immediate value:
    - 12 bits to encode 32 number
    - 11:8 -> 4-bit rotation
    - 7:0 -> 8 bit number
    - the 8-bit number is right rotate around the 32 bits, padded with zeros.
    - 4-bit rotation x 2 = number of bits to rotate
    - some immediate will require more than 1 instructions



Hand Assembly

- I=0:
- Rm register only, no shift
  - add r0, r1, r2:
    - shamt5 = 00000
    - sh = 00
    - rm = 0010

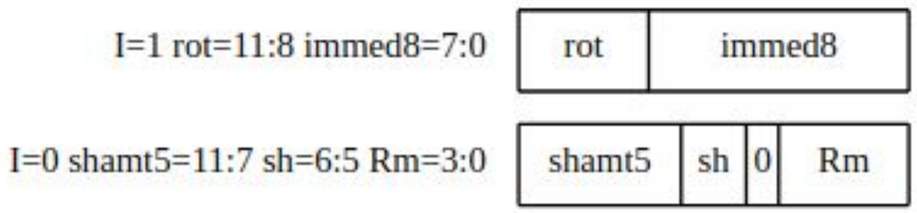


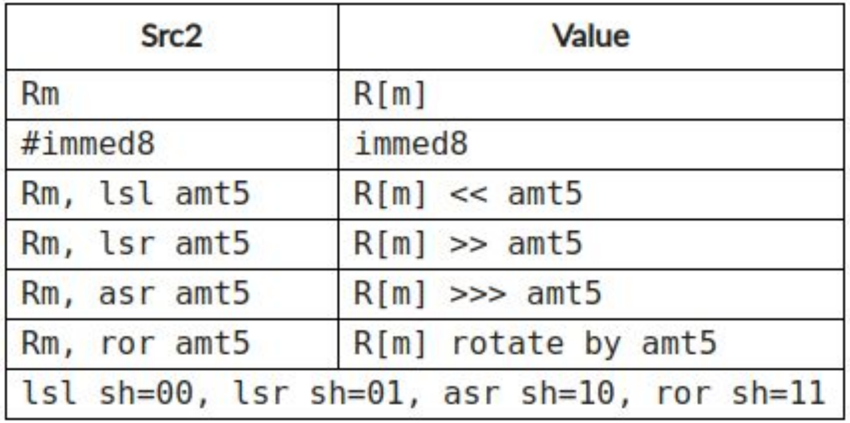


Src2	Value
Rm	R[m]
#immed8	immed8
Rm, lsl amt5	R[m] << amt5
Rm, lsr amt5	R[m] >> amt5
Rm, asr amt5	R[m] >>> amt5
Rm, ror amt5	R[m] rotate by amt5
lsl sh=00, lsr sh=01, asr sh=10, ror sh=11	

Hand Assembly

- I=0:
- Shift instructions can also be written (encoded as mov):
  - lsl Rd, Rm, #shamt5 (sh=00)
  - lsr Rd, Rm, #shamt5 (sh=01)
  - asr Rd, Rm, #shamt5 (sh=10)
  - ror Rd, Rm, #shamt5 (sh=11)
- lsl R0, R9, #7
  - shamt5 = 00111
  - sh = 00
  - Rm = 1001

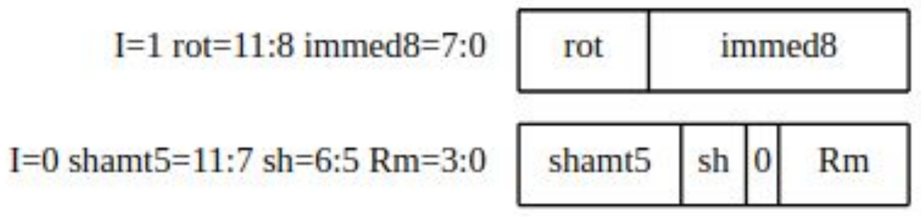


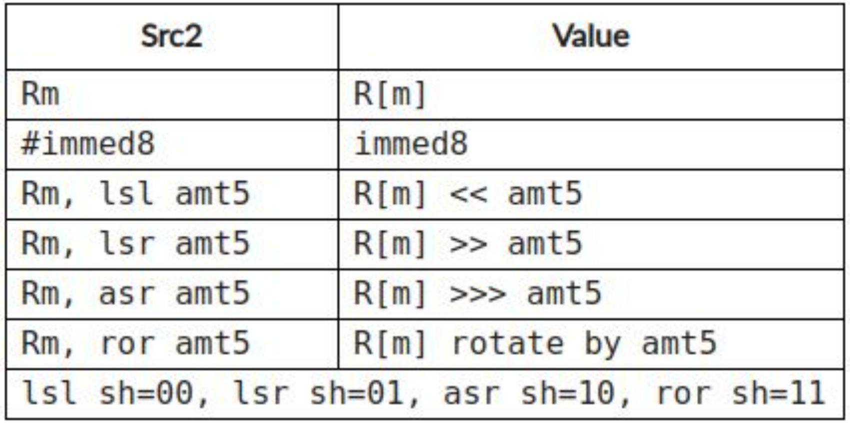


Src2	Value
Rm	R[m]
#immed8	immed8
Rm, lsl amt5	R[m] << amt5
Rm, lsr amt5	R[m] >> amt5
Rm, asr amt5	R[m] >>> amt5
Rm, ror amt5	R[m] rotate by amt5
lsl sh=00, lsr sh=01, asr sh=10, ror sh=11	

Hand Assembly

- I=0:
- Registers values also can be shifted by the amount stored in another register
- register-shifted register addressing mode
  - register (Rm) is shifted by the amount held in a second register (Rs) (11:8 bits, shamt5 msb)
  - least significant 8 bits of Rs are used
  - Rs=0xF001001C
  - shift=0x1C





Src2	Value
Rm	R[m]
#immed8	immed8
Rm, lsl amt5	R[m] << amt5
Rm, lsr amt5	R[m] >> amt5
Rm, asr amt5	R[m] >>> amt5
Rm, ror amt5	R[m] rotate by amt5
lsl sh=00, lsr sh=01, asr sh=10, ror sh=11	

Examples

- Consider the following assembly instruction:

Condition Codes		
cond	Mnemonic	Compare
0000	EQ	== 0
0001	NE	!= 0
0010	HS (u)	>=
0011	LO (u)	<
0100	MI	N
0101	PL	~N
0110	VS	V
0111	VC	~V
1000	HI (u)	>
1001	LS (u)	<=
1010	GE	>= 0
1011	LT	< 0
1100	GT	> 0
1101	LE	<= 0
1110	AL	always
u is unsiged		

Data Processing		
Assembly	RTL	cmd
and Rd, Rn, Src2	R[d] = R[n] & Src2	0000
eor Rd, Rn, Src2	R[d] = R[n] ^ Src2	0001
sub Rd, Rn, Src2	R[d] = R[n] - Src2	0010
rsb Rd, Rn, Src2	R[d] = Src2 - R[n]	0011
add Rd, Rn, Src2	R[d] = R[n] + Src2	0100
adc Rd, Rn, Src2	R[d] = R[n] + Src2 + C	0101
sbc Rd, Rn, Src2	R[d] = R[n] - Src2 - ~C	0110
rsc Rd, Rn, Src2	R[d] = Src2 - R[n] - ~C	0111
tst Rn, Src2	R[n] & Src2, set flags	1000
teq Rn, Src2	R[n] ^ Src2, set flags	1001
cmp Rn, Src2	R[n] - Src2, set flags	1010
cmn Rn, Src2	R[n] + Src2, set flags	1011
orr Rd, Rn, Src2	R[d] = R[n] | Src2	1100
mov Rd, Src2	R[d] = Src2	1101
bic Rd, Rn, Src2	R[d] = R[n] & ~Src2	1110
mvn Rd, Src2	R[d] = ~Src2	1111

```
add r2, r7, r11
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm

Examples

- Consider the following assembly instruction:

```
add r2, r7, r11
```

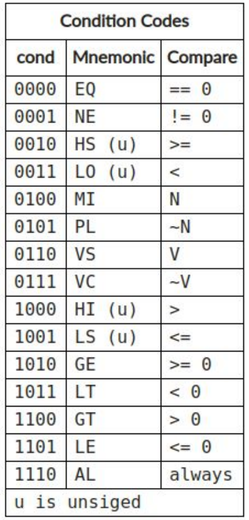
cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
1110	00	0	0100	0	0111	0010	00000	00	0	1011

```
1110 | 0000 | 1000 | 0111 | 0010 | 0000 | 0000 | 1011
E    | 0    | 8    | 7    | 2    | 0    | 0    | B
```

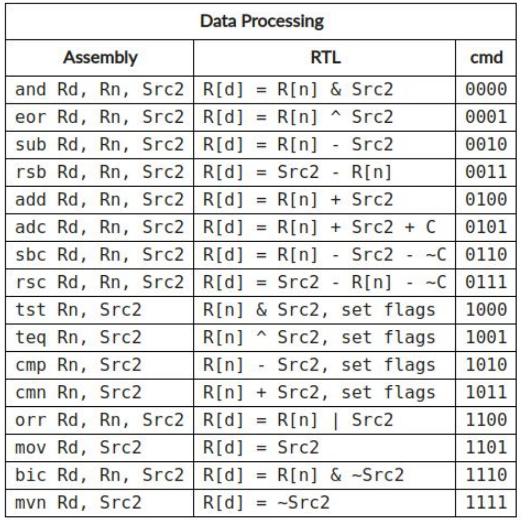
- cond = 1110 = Always, unconditional execution
- In hexadecimal, the instruction is 0xE087200B.

Examples

- Consider the following assembly instruction:



Condition Codes		
cond	Mnemonic	Compare
0000	EQ	== 0
0001	NE	!= 0
0010	HS (u)	>=
0011	LO (u)	<
0100	MI	N
0101	PL	~N
0110	VS	V
0111	VC	~V
1000	HI (u)	>
1001	LS (u)	<=
1010	GE	>= 0
1011	LT	< 0
1100	GT	> 0
1101	LE	<= 0
1110	AL	always
u is unsiged		



Data Processing		
Assembly	RTL	cmd
and Rd, Rn, Src2	R[d] = R[n] & Src2	0000
eor Rd, Rn, Src2	R[d] = R[n] ^ Src2	0001
sub Rd, Rn, Src2	R[d] = R[n] - Src2	0010
rsb Rd, Rn, Src2	R[d] = Src2 - R[n]	0011
add Rd, Rn, Src2	R[d] = R[n] + Src2	0100
adc Rd, Rn, Src2	R[d] = R[n] + Src2 + C	0101
sbc Rd, Rn, Src2	R[d] = R[n] - Src2 - ~C	0110
rsc Rd, Rn, Src2	R[d] = Src2 - R[n] - ~C	0111
tst Rn, Src2	R[n] & Src2, set flags	1000
teq Rn, Src2	R[n] ^ Src2, set flags	1001
cmp Rn, Src2	R[n] - Src2, set flags	1010
cmn Rn, Src2	R[n] + Src2, set flags	1011
orr Rd, Rn, Src2	R[d] = R[n] | Src2	1100
mov Rd, Src2	R[d] = Src2	1101
bic Rd, Rn, Src2	R[d] = R[n] & ~Src2	1110
mvn Rd, Src2	R[d] = ~Src2	1111

```
eors r9,r3,r5
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
	00								0	

Examples

- Consider the following assembly instruction:

```
eors r9,r3,r5
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
1110	00	0	0001	1	0011	1001	00000	00	0	0101

```
1110 | 0000 | 0011 | 0011 | 1001 | 0000 | 0000 | 0101
E    | 0    | 3    | 3    | 9    | 0    | 0    | 5
```

- cond = 1110 = Always, unconditional execution
- In hexadecimal, the instruction is 0xE0339005.

Examples

- Consider the following assembly instruction:

addeq r1, r2, r3

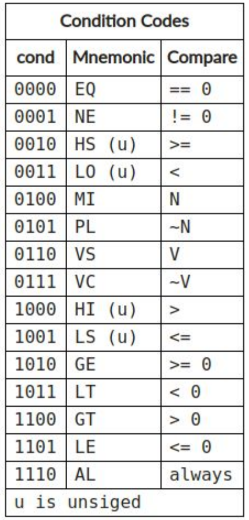
cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
0000	00	0	0100	0	0010	0001	00000	00	0	0011

```
0000 | 0000 | 1000 | 0010 | 0001 | 0000 | 0000 | 0011
0    | 0    | 8    | 2    | 1    | 0    | 0    | 3
```

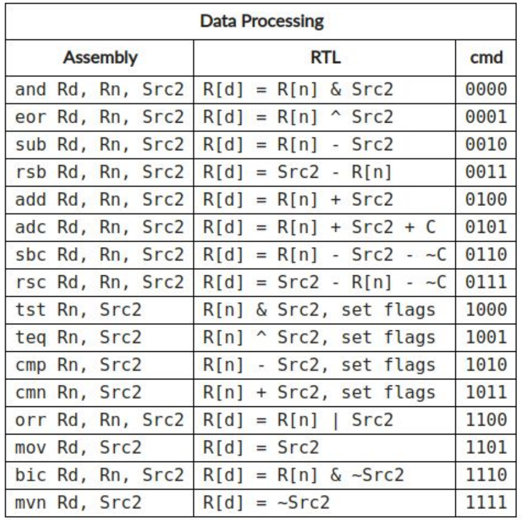
- In hexadecimal, the instruction is 0x00821003.

Examples

- Consider the following assembly instruction:



Condition Codes		
cond	Mnemonic	Compare
0000	EQ	== 0
0001	NE	!= 0
0010	HS (u)	>=
0011	LO (u)	<
0100	MI	N
0101	PL	~N
0110	VS	V
0111	VC	~V
1000	HI (u)	>
1001	LS (u)	<=
1010	GE	>= 0
1011	LT	< 0
1100	GT	> 0
1101	LE	<= 0
1110	AL	always
u is unsiged		



Data Processing		
Assembly	RTL	cmd
and Rd, Rn, Src2	R[d] = R[n] & Src2	0000
eor Rd, Rn, Src2	R[d] = R[n] ^ Src2	0001
sub Rd, Rn, Src2	R[d] = R[n] - Src2	0010
rsb Rd, Rn, Src2	R[d] = Src2 - R[n]	0011
add Rd, Rn, Src2	R[d] = R[n] + Src2	0100
adc Rd, Rn, Src2	R[d] = R[n] + Src2 + C	0101
sbc Rd, Rn, Src2	R[d] = R[n] - Src2 - ~C	0110
rsc Rd, Rn, Src2	R[d] = Src2 - R[n] - ~C	0111
tst Rn, Src2	R[n] & Src2, set flags	1000
teq Rn, Src2	R[n] ^ Src2, set flags	1001
cmp Rn, Src2	R[n] - Src2, set flags	1010
cmn Rn, Src2	R[n] + Src2, set flags	1011
orr Rd, Rn, Src2	R[d] = R[n] | Src2	1100
mov Rd, Src2	R[d] = Src2	1101
bic Rd, Rn, Src2	R[d] = R[n] & ~Src2	1110
mvn Rd, Src2	R[d] = ~Src2	1111

```
mov pc, lr
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
	00								0	

Examples

- Consider the following assembly instruction:

```
mov pc, lr
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
1110	00	0	1101	0	0000	1111	00000	00	0	1110

```
1110 | 0001 | 1010 | 0000 | 1111 | 0000 | 0000 | 1110
E    | 1    | A    | 0    | F    | 0    | 0    | E
```

- pc = r15 = 1111
- lr = r14 = 1110
- cond = 1110 = Always, unconditional execution
- In hexadecimal, the instruction is 0xe1a0f00e.

Examples

- Consider the following assembly instruction:

Condition Codes		
cond	Mnemonic	Compare
0000	EQ	== 0
0001	NE	!= 0
0010	HS (u)	>=
0011	LO (u)	<
0100	MI	N
0101	PL	~N
0110	VS	V
0111	VC	~V
1000	HI (u)	>
1001	LS (u)	<=
1010	GE	>= 0
1011	LT	< 0
1100	GT	> 0
1101	LE	<= 0
1110	AL	always
u is unsiged		

Data Processing		
Assembly	RTL	cmd
and Rd, Rn, Src2	R[d] = R[n] & Src2	0000
eor Rd, Rn, Src2	R[d] = R[n] ^ Src2	0001
sub Rd, Rn, Src2	R[d] = R[n] - Src2	0010
rsb Rd, Rn, Src2	R[d] = Src2 - R[n]	0011
add Rd, Rn, Src2	R[d] = R[n] + Src2	0100
adc Rd, Rn, Src2	R[d] = R[n] + Src2 + C	0101
sbc Rd, Rn, Src2	R[d] = R[n] - Src2 - ~C	0110
rsc Rd, Rn, Src2	R[d] = Src2 - R[n] - ~C	0111
tst Rn, Src2	R[n] & Src2, set flags	1000
teq Rn, Src2	R[n] ^ Src2, set flags	1001
cmp Rn, Src2	R[n] - Src2, set flags	1010
cmn Rn, Src2	R[n] + Src2, set flags	1011
orr Rd, Rn, Src2	R[d] = R[n] | Src2	1100
mov Rd, Src2	R[d] = Src2	1101
bic Rd, Rn, Src2	R[d] = R[n] & ~Src2	1110
mvn Rd, Src2	R[d] = ~Src2	1111

```
lsr R3, R5, #4
(mov in disguise)
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
	00								0	

Examples

- Consider the following assembly instruction:

```
lsr R3, R5, #4
```

cond		I	cmd	S	Rn	Rd	shamt	sh		Rm
1110	00	0	1101	0	0000	0011	00100	01	0	0101

```
1110 | 0001 | 1010 | 0000 | 0011 | 0010 | 0010 | 0101
E    |   1  |  A   |   0  |  3   |   2  |   2  |  5
```

- Shifting are encoded as MOV operation with the immediate as Rm shifted
- In hexadecimal, the instruction is 0xE1A03225.

Examples

- The hand assembly can be checked by creating an assembly file:

```
arm-linux-gnueabi-as -al asm.s

arm-linux-gnueabi-objdump -S a.out
```

- Hexadecimals should match.
- Output order from least significant to most significant byte
- arm-linux-gnueabi-objdump shows 32 bit words.

Repository “arm_emu.sh” correct order of bytes

```
        .arch armv4
        .text
        .align 2
        .global     main
        .arm
main:
        add  r2, r7, r11
        eors r9, r3, r5
        addeq r1, r2, r3
        lsr r0, r5, #4
        mov  pc, lr
        .size main, .-main
```

Questions?

Next: 24 - Controlling an ARM-like data path