

Tutorial 3

Tuesday, April 13, 2021 12:30 PM

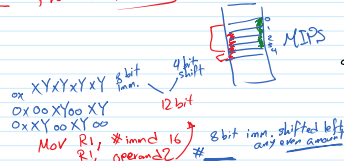
SH
SW
LW
LW

opcode Dest, src1, src2

Q1. What are the differences between the following three instructions?

LDR R0, [R1] LDRH R0, [R1] LDRB R0, [R1]
LDR load one word from the memory 4 bytes
LDRH load half word (2 bytes) unsigned from the memory
LDRB load one byte unsigned from the memory

LDRSH
LDRSB 16 bits → 32 bits



Add {cond} {S} {Rd}, R1, operand2

Mov R0, #10
Add R1, R1, R0
R1 = R1 + R0
Add R1, R0
R1 = R1 + R0

6/3 = 2
5/3 = 1.667
Q R
1 2
1.5
(5+2)/3 = 2

Q2. Translate the following program into C:

```
LDR R3, #N      R3 = &N
LDR R1, [R3]     R1 = Mem[R3] = N
MOV R0, #3003    R0 = 3003
MUL R1, R0, R1    R1 = R0 * R1 = 3003 * Mem[R3]
ADD R1, R1, #512  R1 = 3003 * Mem[R3] + 512
LDR R2, #M       R2 = M
LDR R2, [R1, #10] R2 = Mem[R1 + 10]
STR R0, [R2]      Mem[R2] = (3003 * Mem[R3] + 512) / 1024
```

if (A > max) max = A;

CMP R1, R2
MOVGT R2, R1

CMP, TEST, --
set flags
CMP R1, R2
BLE SKIP
MOV R2, R1
SKIP

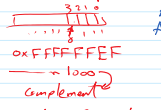
EQ
NE
LS
LO
GT
≥ unsigned
≤
≥ signed

LDR R2, [R1]
R2 = Mem[R1]

Adds R1, R1, R0
R1 = R1 + R0

Q3. Embedded systems always require the user to manipulate bits in registers or variables. Given an integer variable a, write two code fragments in C. The first should set bit 3 of a. The second should clear bit 3 of a. In both cases, the remaining bits should be unmodified.

Clear ⇒ And 0
Set ⇒ OR 1
First fragment:
a = a | (1 << 3);
Second fragment:
a = a & ~(1 << 3);



LDR R2, [R1]
LDR R2, [R1, #4] R2 = Mem[R1 + 4] R1 unchanged
LDR R2, [R1, #4] R2 = Mem[R1 + 4], R1 = R1 + 4
LDR R2, [R1, #4] R2 = Mem[R1 + 4], R1 = R1 + 4
LDR R2, [R1, R0] R2 = Mem[R1 + R0]
LDR R2, [R1, R0, LSL #2] R2 = Mem[R1 + 4 * R0]

Q4. Develop a sequence of instructions that sets the rightmost four bits of R3, clears the leftmost three bits of R3, and inverts bit positions 7, 8 and 9 of R3. Assuming that R3 is 16-bit register.

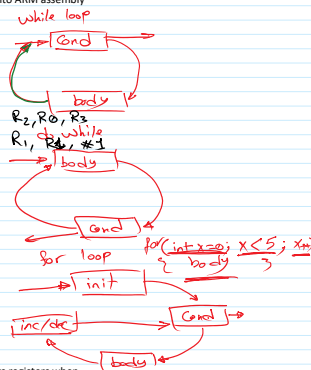
```
ORR R3, R3, #0xF      (0x0F 0x000F) 1110
AND R3, R3, #0xFF     (0xFF 0x0000) 0x1FFF
EOR R3, R3, #0x80     (0x80 0x0000) 0x1FFF
```

Result = A - B
Z = Result == 0

CMP, TEST, --
Adds R1, R0, R1
sub

Q5. Translate the below C code for counting the number of occurrence of ones in R0 into ARM assembly code, using the registers indicated by the variable names.

```
r3 = 1;
r1 = 0;
while (r3 != 0) {
    if ((r0 & 1) == 0) {
        r1 = r1 + 1;
    }
    r3 = r3 >> 1;
}
BAL skip
Exit
```



Q6. Explain what an ARM processor accomplishes in terms of accessing and changing its registers when it executes a BEQ instruction.

BEQ check PSR flag Z
if Z = 1, update PC register to the label address
else PC increment automatically to execute the next instruction.

Q7. For the below ARM assembly code, trace the values as it executes that will be placed into the registers PC, R0, R1, and R2.

```
PC=4 0    MOV R0, #0    ; at addr 0
8 4    MOV R1, #5
12 8    ADD R0, R0, R1
16 12    SUB R1, R1, #1
20 16    CMP R0, #10
24 20    MOVGT PC, #28
28 24    MOV PC, #8
32 28    MOV R2, #1
36 32    halt    B    halt    ; at address
```

PC 0, 4, 8, 12, 16, 20, 24, 8, 12, 16, 20, 24, 8, 12, 16, 20, 28, 32, 32, 32, 32.

R0 0 5 9 12

R1 5 4 3 2

R2 1

PC points to the instruction to be fetched.