EECS 370 -- ARM Examples

Registers

- General Purpose:
    - R0-R12 General purpose registers
- Special Purpose:
    - R13 Stack Pointer (SP)
    - R14 Link Register (LR)
    - R15 Program Counter (PC)

- Special:
    - PSR Program Status Register

ARM Address Modes

- Register Indirect Addressing:

```
LDR r0,[r1]
        If r1 = 0x100
        Set r0 = mem[0x100]
STR r0,[r1]
        If r1 = 0x100
        Set mem[0x100] = r0
```

- Base+displacement Addressing
    - Displacement can be either a register or a 12-bit immediate.
    - Displacement can be added or subtracted.
    - Register displacement can be shifted/rotated (5-bit), immediate cannot.

- Pre-indexed

```
strr3, [r4,#-34]!
        MEM[r4 + -34] = r3; r4 = r4 – 34;
```

- Post-indexed

```
strr3, [r4], #-34
        MEM[r4] = r3; r4 = r4 – 34;
```

Single Register Load and Store Instructions

- LDR: load word

```
LDR r0, [r1,#1000]; retrieve a word(32 bits) from address (r1+1000)
```

- LDRH: load halfword unsigned
  LDRH r0, [r1,#1000]; retrieve a halfword(16 bits) unsigned from address (r1+1000)
- LDRB: load byte unsigned
  LDRB r0, [r1,#1000]; retrieve a byte(8 bits) from address (r1+1000)
- LDRSH: load halfword sign extend
  LDRSH r0, [r1,#1000]; retrieve a halfword(16 bits) sign extend from address (r1+1000)
- LDRSB: load byte sign extend
  LDRSB r0, [r1,#100]; retrieve a byte(8 bits) sign extend from address (r1+100)
- STR: store word
  STR r0, [r1,#1000]; store all 32 bits of r0 to MEM[r1+1000]
- STRH: store half-word
  STRH r0, [r1,#1000]; store 16 least significant bits of r0 to MEM[r1+1000]
- STRB: store byte
  STRB r0, [r1,#1000]; store 8 least significant bits of r0 to MEM[r1+1000]

ARM Arithmetic Instructions

- ADD: add
  ADD r0,r1,r2; r0 = r1+r2*
- SUB: subtract
  SUB r0, r1,r2; r0 = r1-r2*
- RSB: reverse subtract
  RSB r0,r1,r2; r0 = r2-r1*
- MUL: multiply
  MUL r0,r1,r2; r0 = r1*r2
- AND: Bit-wise and
  AND r0,r1,r2; r0 = r1 & r2*
- ORR: Bit-wise or
  ORR r0,r1,r2; r0 = r1 | r2*
- EOR: Bit-wise exclusive-or
  EOR r0,r1,r2; r0 = r1 XOR r2*
- BIC: bit clear
  BIC r0,r1,r2; r0 = r1 & NOT(r2*)

ARM Move Instructions

Register-to-register operations

- MOV: move
  MOV r0,r1; r0 = r1*

- MVN: move (negated)

  MVN r0,r1; r0 = NOT(r1*)

ARM Flexible Operand2

The last operand of most ARM instructions can be called operand2 (marked above with an r2*). It is flexible and has different formats:

- Immediate Value(8-bit)

  mov r0, #13 // r0 = 13; - Register
  mov r0, r1 // r0 = r1;

- Register with Shift/Rotate
  - Shift value can be a constant or a register.
  - If shift value is a constant, the range will depend on the instruction.
  - If a register holds the shift value, only least significant byte will be used.

- Logical Shift Left
  - Constant shift value will be in the range 0-31.
  - Fill bits with zeros.
    - 0b0011 logical shift left by 2 -> 0b1100

```
movr0, r1, lsl r2 //r0=r1<<(r2&0xFF);
movr0, r1, lsl #2 //r0=r1<<2;
```

- Logical Shift Right
  - Constant shift value will be in the range 1-32.
  - Fill bits with zeros.
    - 0b1001 logical shift right by 2 -> 0b0010

```
movr0, r1, lsr r2      //r0=r1>>(r2&0xFF); r1 declared as unsigned;
movr0, r1, lsr #2      //r0=r1>>2; r1 declared as unsigned;
```

- Arithmetic Shift Right
  - Constant shift value will be in the range 1-32.
  - Fill bits with sign bits.
    - 0b1001 arithmetic shift right by 2 -> 0b1110

```
movr0, r1, asr r2      //r0=r1>>(r2&0xFF); r1 declared as signed;
movr0, r1, asr #2      //r0=r1>>2; r1 declared as signed;
```

- Rotate Right
  - Constant shift value will be in the range 1-31.

```
mov r0, r1, ror #4 // If r1 == 0xABCD EF01, then r0 = 0x1ABC DEF0;
```

ARM Comparison Instructions

Most arithmetic/logic instructions with a S suffix and compare instructions can set the Program Status Register (PSR). Four flags can be set:

- N: set if the result is negative.
- Z: set if the result is zero.
- C: set if last addition/subtraction had a carry/borrow out of bit 31
- V: set if last addition/subtraction produced an overflow

- Compare
  ```
  cmp r0, r1      // Compute r0 – r1 and set PSR;
  ```

- Negated Compare
  ```
  cmn r0, r1      // Compute r0 + r1 and set PSR;
  ```

ARM Condition Code

All instructions can have any one of the condition codes after the instruction code, which means it will be executed when such condition holds. List of condition codes:

eq:	EQual	Z==1
ne:	Not Equal	Z==0
ge:	Greater than/Equal	N==V
lt:	Less Than	N!=V
gt:	Greater Than	Z==0&&N==V
le:	Less than/Equal	Z==1||N!=V
cs:	unsigned higher/Same	C==1
cc:	unsigned lower	C==0
mi:	negative	N==1
pl:	positive/Zero	N==0
vs:	oVerflow Set	V==1
vc:	no oVerflow Clear	V==0
hi:	unsigned HIgher	C==1&&Z==0
ls:	unsigned Lower/Same	C==0||Z==1
al:	any/ALways	*

ARM Flow Control

Branches use PSR to decide if it is taken or not. Branch Instruction:

- Branch

```
b{cond} #100        // Ifconditionholds, target address = PC + 8 + 4 * 100;
b{cond} label       // Ifconditionholds, branch to this label;
```

- Branch and Link
    - Used for subroutine call, return address will be saved in LR(r14).

```
bl{cond} label      // If condition holds, save PC of next instruction and
                    // branch to the label;
```

Example:

```
bl foo
...
foo: …
mov  r15, r14  // Return;
```