50.002 COMPUTATIONAL STRUCTURES

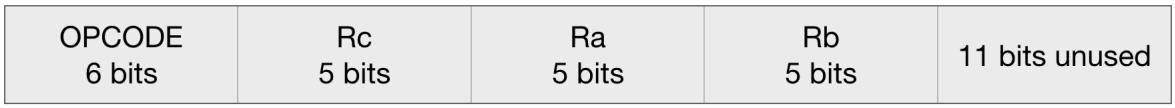
INFORMATION SYSTEMS TECHNOLOGY AND DESIGN

Explanations on Beta Instructions

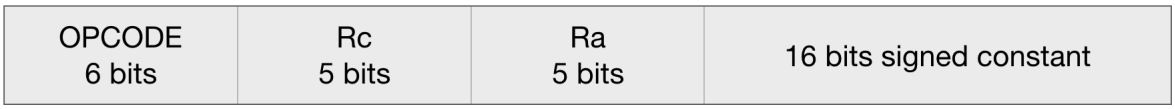
Natalie Agus (Fall 2018)

1 Introduction

In this document, I try to give examples and explain some of the more confusing β instructions as humanly as possible. Recall that there's two types of Opcodes,



Type 1



Type 2

Figure 1

We can divide the instruction sets (basically assembly language) into three groups:

1. Operate Class Type 1 : `ADD, SUB, MUL, DIV, AND, OR, XOR, CMPEQ, CMPLT CMPLE, SHL, SHR, SRA`
2. Operate Class Type 2 : `ADD, SUB, MUL, DIV, AND, OR, XOR, CMPEQ, CMPLT CMPLE, SHL, SHR, SRA`
3. Other : `LD, ST, JMP, BEQ/BF, BNE/BT, LDR`

Other useful stuff: a (dot) `.  = 0xZZZZ` **means that the *next* line of instruction is written at RAM address** `0xZZZZ`**. We can also "name" a memory location such**

as `result = 0xABCD`**. See notes on *Stacks* to learn more. A PC on the other hand, always starts to execute from memory address** `0x0000`**.**

2 PC ← PC + 4

Notice that at each operation, you always increase the PC by 1 word length (4 bytes) when you execute the operation. For example:

1. DIV(Ra, Rb, Rc) operation is PC ← PC + 4, and then Reg[Rc] ← Reg[Ra] / Reg[Rb]
2. ADD(Ra, Rb, Rc) operation is PC ← PC + 4, and then Reg[Rc] ← Reg[Ra] + Reg[Rb]
3. JMP(Ra, Rc) operation is PC ← PC + 4, and then EA ← Reg[Ra] & 0xFFFFFFFC, Reg[Rc] ← PC, PC ← EA

This simply means you set the PC ready to execute the **next instruction in the next memory line** because you have already executed this current line of instruction.

3 Operate Class Type 1

In this section, we are going to go through two selected operate classes of Type 1: `SHR, SRA` and one of the compares: `CMPEQ`.

1. `SHR(Ra, Rb, Rc)` : means that you **logically shift** the content of register RA to the right by the amount specified by the **index** of Rb and store the shifted content in Rc.
   (a) For example, given `SHR(4, 3, 2)`
   (b) And 32-bit content of R4 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000
   (c) And that PC is at address 0x0000 0004 for this instruction
   (d) This means Ra = 4, Rb = 3, Rc = 2
   (e) We can translate the instruction into 32-bit instruction of Type 1: 101101 00010 00100 00011 00000000000.
   (f) This means we logically shift (pad with zeroes) the content of R4 by 3 bits to the right (because the index of Rb is 3)
   (g) So we will store: 0001 0001 0001 0001 0001 0001 0001 0001 inside R2.
   (h) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

2. `SHA(Ra, Rb, Rc)` : means that you **arithmetically shift** the content of register RA to the right by the amount specified by the **index** of Rb and store the shifted content in Rc. The difference between logical and arithmetical shift is that in arithmetical shift, we preserve the sign bit.

   (a) For example, given `SHA(4, 3, 2)`

   (b) And 32-bit content of R4 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000

   (c) And that PC is at address 0x0000 0004 for this instruction

   (d) This means Ra = 4, Rb = 3, Rc = 2

   (e) We can translate the instruction into 32-bit instruction of Type 1: 101101 00010 00100 00011 00000000000.

   (f) This means we arithmetically shift (pad with the whatever MSB is there in the first place) the content of R4 by 3 bits to the right (because the index of Rb is 3)

   (g) So we will store: 1111 0001 0001 0001 0001 0001 0001 0001 inside R2.

   (h) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

3. `CMPEQ(Ra,Rb,Rc)` : means that you compare if the content in Ra is equal to the content in Rb, and store 1 in Rc if true, or 0 in Rc if false.

   (a) For example, given `CMPEQ(15, 14, 8)`

   (b) And 32-bit content of R15 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000

   (c) And 32-bit content of R14 (Rb) : 1000 1000 1000 1000 1000 1000 1111 1000

   (d) And that PC is at address 0x0000 0004 for this instruction

   (e) This means Ra = 15, Rb = 14, Rc = 8

   (f) We can translate the instruction into 32-bit instruction of Type 1: 100100 01000 01111 01110 00000000000.

   (g) Of course the content of R15 is not the same as R14 in this case,

   (h) So we will store: 0000 0000 0000 0000 0000 0000 0000 0000 inside R8.

   (i) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

4 Operate Class Type 2

In this section, we are going to go through two selected operate classes of Type 2: `SHRC`, `SRAC` and one of the compares: `CMPEQC`.

1. `SHRC(Ra, 16-bit signed literal, Rc)` : means that you **logically shift** the content of register RA to the right by the amount specified by the **5 least significant bits of the literal** and store the shifted content in Rc.
   (a) For example, given `SHRC(4, 3, 2)`
   (b) And 32-bit content of R4 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000
   (c) And that PC is at address 0x0000 0004 for this instruction
   (d) This means Ra = 4, literal = 3, Rc = 2
   (e) We can translate the instruction into 32-bit instruction of Type 2: 111101 00010 00100 0000 0000 0000 0011.
   (f) This operation will look at the 5 least significant bits of the literal : 00011 to shift the content of R4
   (g) This means we logically shift (pad with zeroes) the content of R4 by 3 bits to the right
   (h) So we will store: 0001 0001 0001 0001 0001 0001 0001 0001 inside R2.
   (i) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

2. `SHAC(Ra, 16-bit signed literal, Rc)` : means that you **arithmetically shift** the content of register RA to the right by the amount specified by the **5 least significant bits of the literal** and store the shifted content in Rc. The difference between logical and arithmetical shift is that in arithmetical shift, we preserve the sign bit.
   (a) For example, given `SHAC(4, 3, 2)`
   (b) And 32-bit content of R4 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000
   (c) And that PC is at address 0x0000 0004 for this instruction
   (d) This means Ra = 4, literal = 3, Rc = 2
   (e) We can translate the instruction into 32-bit instruction of Type 2: 111110 00010 00100 0000 0000 0000 0011.
   (f) This operation will look at the 5 least significant bits of the literal : 00011 to shift the content of R4
   (g) This means we arithmetically shift (pad with the whatever MSB is there in the first place) the content of R4 by 3 bits to the right
   (h) So we will store: 1111 0001 0001 0001 0001 0001 0001 0001 inside R2.
   (i) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

3. `CMPEQ(Ra,16-bit signed literal,Rc)` : means that you compare if the content in Ra is equal to the literal, and store 1 in Rc if true, or 0 in Rc if false.

(a) For example, given `CMPEQ(15, -32768, 8)`

(b) And 32-bit content of R15 (Ra) : 1000 1000 1000 1000 1000 1000 1000 1000

(c) And that PC is at address 0x0000 0004 for this instruction

(d) This means Ra = 15, literal = -32768, Rc = 8

(e) We can translate the instruction into 32-bit instruction of Type 2: 110100 01000 01111 1000 0000 0000 0000.

(f) Of course the content of R15 is not the same as -32768 in this case,

(g) So we will store: 0000 0000 0000 0000 0000 0000 0000 0000 inside R8.

(h) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

5 Other operations

In this section we are going to go through examples of all of these 'other' operations, meaning the operations that involve shifting the PC. The format of all of these 'other' operations are Type 2.

1. `LD(Ra, 16-bit signed literal, Rc)` : means that you load the **content** of the memory with address : content of Ra added plus the literal into Rc.

   (a) For example, given `LD(R3, 8, R9)`

   (b) And 32-bit content of R3 is : 0x0000 0010 (wrote it in hex for convenience purpose)

   (c) And that PC is at address 0x0000 0004 for this instruction

   (d) This means Ra = 3, literal = 8, Rc = 9

   (e) We can translate the instruction into 32-bit instruction of Type 2: 011000 01001 00011 0000 0000 0000 1000

   (f) In hex, the literal is 0x0008

   (g) This means we need to first compute the new address, which is the content of R3 added by 0x0008 = 0x0000 0018

   (h) Given that the 32-bit content of the memory at address 0x0000 0018, i.e: Mem[0x0000 0018] = 0x1234 5678,

   (i) We store this 32-bit content: 0x1234 5678 at R9

   (j) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

2. `ST(Rc, 16-bit signed literal, Ra)` : means that you store the content of register Rc to the memory with address : content of Ra plus the literal.

   (a) For example, given `ST(R9, 8, R3)`

(b) And 32-bit content of R3 is : 0x0000 0010 (wrote it in hex for convenience purpose)

(c) And 32-bit content of R9 is : 0x1234 ABCD

(d) And that PC is at address 0x0000 0004 for this instruction

(e) This means Ra = 3, literal = 8, Rc = 9

(f) We can translate the instruction into 32-bit instruction of Type 2: 011001 01001 00011 0000 0000 0000 1000

(g) In hex, the literal is 0x0008

(h) This means we need to first compute the new address, which is the content of R3 added by 0x0008 = 0x0000 0018

(i) And then we store the content of R9, which is 0x1234 ABCD to the content of the memory with address 0x0000 0018, i.e: Mem[0x0000 0018] = 0x1234 ABCD

(j) And increase PC by 4 bytes to address 0x0000 0008 to execute the (next) instruction at this new address

3. `JMP(Ra, Rc)` : means that you store the address of PC + 4 at Rc, and then move PC to the address as specified in the content of Ra.

   (a) For example, given JMP(R4, R10)

   (b) And that PC is at address 0x0000 0004 for this instruction

   (c) And 32-bit content of R4 is : 0x0000 100C

   (d) We can translate the instruction into 32-bit instruction of Type 2: 011011 01010 00100 0000 0000 0000 0000

   (e) PC + 4 is 0x0000 0008, we store this in R10, so now the content of R10 becomes 0x0000 0008

   (f) Then the PC is now set to the content of R4, which is 0x0000 100C

   (g) Meaning that we will now execute the 32-bit instruction that is present as the content of memory address 0x0000 100C

4. `BEQ/BF(Ra, label, Rc)` : means that you store the address of PC + 4 at Rc, and then move the PC to the address of the label if the content of Ra is equal to zero.

   (a) For example, given BEQ(R4, f, R10) (or equivalently BF(R4, f, R10))

   (b) And that the address of instruction f in the memory [1] is 0x0000 1004 (f is another instruction in the memory, which obviously has a label f)

   (c) And that PC is at address 0x0000 0004 for this instruction

   (d) And 32-bit content of R4 is : 0x0000 0000

[1] Recall the memory can hold both data and instructions

(e) To compute the literal for this instruction, we need **three steps**. First, (1) compute how many bytes difference are there between the address of f and the current PC: which is 0x0000 1004 - 0x0000 0004 = 0x0000 1000 = 16^3 = 4096 bytes. And then we need to (2) convert this into word unit: 4096 / 4 = 1024 words (1 word = 4 bytes). Finally, we need to (3) subtract by 1 : 1024 - 1 = 1023. So the literal for this instruction is 1023. In 16 bits, 1023 is 0000 0011 1111 1111.

(f) Therefore the 32-bit instruction of Type 2 for this example is: 011101 01010 00010 0000 0011 1111 1111

(g) PC + 4 is 0x0000 0008, and this is stored in R10

(h) We compare now the content of R4 on whether it is equal to zero

(i) Since the content of R4 is equal to zero, PC is then moved to 0x0000 0004 + 4 + 4*1023 = 0x0000 0008 + 0x0000 0FFC = 0x0000 1004 (which is just basically the address of that instruction f)

(j) *If the content of R4 is not equal to zero, then PC ← PC + 4 and will end up at 0x0000 0008 to execute the next line instruction in the memory*

5. `BNE/BT(Ra, label, Rc)` : means that you store the address of PC + 4 at Rc, and then move the PC to the address of the label if the content of Ra is **not equal** to zero.

(a) For example, given BNE(R4, f, R10) (or equivalently BT(R4, f, R10))

(b) And that the address of instruction f in the memory [2] is 0x0000 1004 (f is another instruction in the memory, which obviously has a label f)

(c) And that PC is at address 0x0000 0004 for this instruction

(d) And 32-bit content of R4 is : 0x0000 1010

(e) Similarly, to compute the literal for this instruction, we need **three steps**. First, (1) compute how many bytes difference are there between the address of f and the current PC: which is 0x0000 1004 - 0x0000 0004 = 0x0000 1000 = 16^3 = 4096 bytes. And then we need to (2) convert this into word unit: 4096 / 4 = 1024 words (1 word = 4 bytes). Finally, we need to (3) subtract by 1 : 1024 - 1 = 1023. So the literal for this instruction is 1023. In 16 bits, 1023 is 0000 0011 1111 1111.

(f) Therefore the 32-bit instruction of Type 2 for this example is: 011110 01010 00010 0000 0011 1111 1111

(g) PC + 4 is 0x0000 0008, and this is stored in R10

(h) We compare now the content of R4 on whether it is equal to zero

(i) Since the content of R4 is **not equal** to zero, PC is then moved to 0x0000 0004 + 4 + 4*1023 = 0x0000 0008 + 0x0000 0FFC = 0x0000 1004 (which is just basically the address of that instruction f)

[2] Recall the memory can hold both data and instructions

(j) *If the content of R4 is equal to zero, then PC ← PC + 4 and will end up at 0x0000 0008 to execute the next line instruction in the memory*

6. `LDR(label, Rc)` : means that you load the content of the memory which address is the address of the 'label', to Rc.

   (a) For example, given LDR(f, R16)

   (b) And that the address of instruction f in the memory [3] is 0x0000 0008(f is another instruction in the memory, which obviously has a label f)

   (c) And that PC is at address 0x0000 1234 for this instruction

   (d) Similarly, to compute the literal for this instruction, we need **three steps**. First, (1) compute how many bytes difference are there between the address of f and the current PC: which is 0x0000 0008 - 0x0000 1234 = 0xFFFF EDD4 = -4652 bytes. And then we need to (2) convert this into word unit: -4652 / 4 = -2326 words (1 word = 4 bytes). Finally, we need to (3) subtract by 1 : -2326 - 1 = -2327. So the literal for this instruction is -2327. In 16 bits, -2327 is 1111 0110 1110 1001.

   (e) **Notice how the literal is negative value, meaning that the address of 'f' is before PC**

   (f) Therefore the 32-bit instruction of Type 2 for this example is: 011111 10000 00000 1111 0110 1110 1001

   (g) We then load the content of memory with address of the 'label', i.e: Mem[0x0000 0008] to Rc

   (h) And finally we move PC to the next line: PC ← PC + 4, and PC ends up at address 0x0000 1238.

[3] Recall the memory can hold both data and instructions

6 PC Supervisor Bit

The high bit of the PC (MSB) is dedicated as the 'Supervisor' bit. The instruction fetch and LDR instruction ignore this bit, treating it as if it were zero. The JMP instruction is allowed to clear the Supervisor bit but not set it, and no other instructions may have any effect on it. Only exceptions cause the Supervisor bit to become set.

7 Reserved Registers

By convention, the content of the following registers are by default set to be:

Register	Symbol	Usage
R31	R31	**Content** is always zero
R30	XP	Exception pointer
R29	SP	Stack Pointer
R28	LP	Linkage Pointer
R27	BP	Base of Frame Pointer

Table 1: Reserved Registers

8 Macros

Some instructions can be written in a more convenient way, as follows:

Macro	Definition
BEQ(Ra, label)	BEQ(Ra, label, R31)
BF(Ra, label)	BF(Ra, label, R31)
BNE(Ra, label)	BNE(Ra, label, R31)
BT(Ra, label)	BT(Ra, label, R31)
BR(label, Rc)	BEQ(R31, label, Rc)
BR(label)	BR(label, R31)
JMP(Ra)	JMP(Ra, R31)
LD(label, Rc)	LD(R31, label, Rc)
ST(Rc, label)	ST(Rc, label, R31)
MOVE(Ra, Rc)	ADD(Ra, R31, Rc)
CMOVE(c, Rc)	ADDC(R31, c, Rc)
PUSH(Ra)	ADDC(SP, 4, SP), then ST(Ra, -4, SP)
POP(Rc)	LD(SP, -4, Rc), then SUBC(SP, 4, SP)
ALLOCATE(k)	ADDC(SP, 4*k, SP)
DEALLOCATE(k)	SUBC(SP, 4*k, SP)

Table 2: Convenient Macros