CSCE230301 - Comp Org.& Assmbly Lang Prog (2020 Summer)

Project 1

Andrew Nady

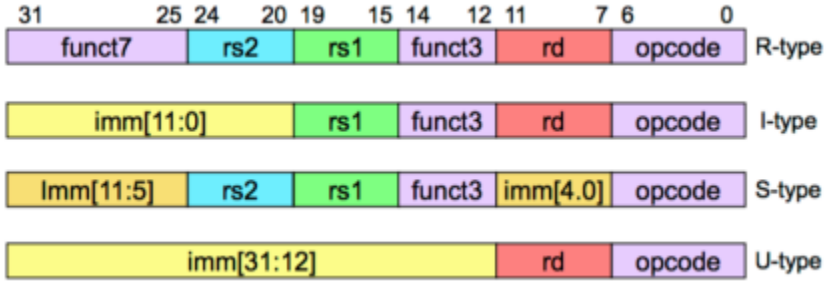
Mahmoud Elshenawy

Mohamed Basuony

Table of Contents

1) Understanding Instructions...……3
……
2) Handling each type…….……4
……
3) Alternative Ideas ……6
……
3) Confirming tests ……7
……

Understanding instructions:



In order to start disassembling, first, each word needs to be dissected into smaller groups of bits. These bits will be opcodes, rd, rs, rs2, func7 and 3, and immediate locations. This process was done using shifting and masking to isolate the bits that represent each segment of the instruction word.

```
opcode = instWord & 0x0000007F;
rd = (instWord >> 7) & 0x0000001F;
funct3 = (instWord >> 12) & 0x00000007;
rs1 = (instWord >> 15) & 0x0000001F;
rs2 = (instWord >> 20) & 0x0000001F;
```

In the following code segment, the first 7 bits of the instruction word were stored in the variable named opcode, then we shifted the instruction to the right and used and to isolate the bits that represent the destination register in the variable rd. Then we shifted the word 12 bits to the right and retrieved the bits that represent fun3 and stored it in fun3. The same operation was repeated to store rs1 and rs2.

However, if the instruction word is a type where it stores and immediate, a variable that stores the immediate value was needed. Not all instructions have the immediate in the same location. For instance, I-type instruction has the immediate starting from bit number 20; S-type has the immediate starting from bit 7 to 11 and then the rest of the immediate is in bit number 25. And in

U-type the immediate takes most of the word as it starts from bit 12. So how did we handle this? We handled this by creating variables for each type of instruction that has an immediate.

For I-type instructions, the immediate is located starting from bit number 20 to the end. Therefore, we created an integer variable to store the mask and store the value of these bits.

```
I_imm = (instWord >> 20) & 0x0000000FFF;
```

For S-type instructions, the immediate first 5 bits are stored in 7-11 bits, so they are retrieved by shifting the word to the right and creating a mask for them. Then they are added to the remaining bits at location 25-31 using bitwise or.

```
S_imm = ((instWord >> 7) & 0x0000001F) |
        (((instWord >> 25) & 0x00000007F)) << 5;
```

The same thing was done with other instructions with different types with respect to the location of the bits that represent the value of their immediates.

```
B_imm = 0x0 |
        ((instWord >> 8) & 0x0000000F) << 1 |
        ((instWord >> 25) & 0x00000003F) << 5 |
        ((instWord >> 7) & 0x000000001) << 11 |
        ((instWord >> 31) & 0x000000001) << 12;

J_imm = ((instWord >> 21) & 0x00000003FF) |
        ((instWord >> 20) & 0x000001) << 10 |
        ((instWord >> 12) & 0x0000000FF) << 11 |
        ((instWord >> 31) & 0x00000001) << 19;
```

Handling each type:

R-Type instructions:

The R-type instructions are differentiated from each other using fun3. However, some instructions have the same fun3 value. Therefore, fun7 is used to differentiate between them. In

our code, we used a series of conditional statements. These statements check the value of func3 and executes prints out the command responding to its value. When there are two commands with the same value, another conditional statement is placed inside to check the value of fun7 and print the command corresponding to its value.

I-Type and S-Type instructions:

Their instructions use only fun3 to differentiate between the commands. Each command has a unique fun3 value associated with it. A series of conditional statements were used to handle the value of fun3 and print the corresponding command.

SB-type instructions:

Since our group was given the extra task of handling the labels, we handled SB-types in a different way. We created an array that stores the immediates that represent the values of the label's offset and checks the rest of the code for any reoccurrence. If it found the same immediate again, it will assign the same label name as the one found in the first occurrence. Fun3 determines which command will be printed.

ecall: The opcode for it is 73 and it is printed when this opcode is found.

Jalr: The opcode for it is 67 and it is printed when this opcode is found.

Jal:

The Jal instruction was handled in a special because it contains a label to jump to. Offset-j variable was used to determine the place of the pc after the label was called. The pc is incremented by J-immediate twice because it represents half a word. If the J-immediate is less than the pc (the label is in a previous command) then offset will be less than pc. If J-immediate is greater than the pc, the offset will be greater than pc. In order to know which line to print the label in a variable named as was created. Its value is shown in this picture. It has +4 because it wants to increment a line to the pc. And It is divided by 4 to determine the number of lines we need to jump to.

```
offset_j=(pc + J_imm+J_imm);
```

```
int as=(((pc- offset_j+4))/4);
```

U-type instructions:

They were handled using their special 2 opcodes. If the opcode was equal to 17 it prints AUIPC command, and if it was equal to 37, it printed LUI.

Alternative Ideas:

While thinking of a way to manipulate the labels in the code, we thought of using a map that stores words and the equivalent instructions in front of it. By creating a map we can edit it as much as we want and print it after we are finished with disassembling all the binary code. However, we found a much easier way to do it by manipulating the cout line. The variable as is

```
cout<<"\033["<<as<<"A"<<"loop"<<l-1<<":"<<"\n";
```

The number of lines that we are supposed to go back to. It is calculated by the place of the pc- j-immediate /4.

Confirming tests:

We used the binary files posted on blackboard and also converted our answer to problem 6 and 1 from the assignment to binary using RARS to

```
aucs-mbp:project aucuser$ ./a.out samples/parr/parr.bin
main:
                ADDI    x13, x0, 1024
                ADDI    x5, x0, 0
                ADDI    x6, x0, 4
        loop1:  BGE     x5, x6,  label1
                ADDI    x17, x0, 5
                ecall
                SLLi    x7, x5, 2
                ADD     x7, x7, x13
                SW      x10, 0(x7)
                ADDI    x5, x5, 1
                JAL     x0, loop1
label1:         ADDI    x5, x0, 0
                ADDI    x6, x0, 3
loop2:          BLT     x6, x5,  label2
                SLLi    x7, x6, 2
                ADD     x7, x7, x13
                LW      x10, 0(x7)
                ADDI    x17, x0, 1
                ecall
                ADDI    x6, x6, -1
                JAL     x0, loop2
label2:         ADDI    x17, x0, 10
                ecall
aucs-mbp:project aucuser$
```

confirm that the disassembler is working properly.

```
aucs-mbp:project aucuser$ ./a.out samples/fib/fib.bin
main:
                ADDI    x2, x0, 1024
                SLLi    x2, x2, 3
                ADDI    x17, x0, 5
                ecall
                JAL     x1, return function1
                ADDI    x10, x17, 0
                ADDI    x17, x0, 1
                ecall
                ADDI    x17, x0, 10
                ecall
return function1:
loop2:  loop1:  ADDI    x5, x0, 2
                BGEU    x10, x5,  label1
                ADDI    x17, x10, 0
                JALR    x0, x1, 0
label1:         ADDI    x2, x2, -12
                SW      x10, 0(x2)
                SW      x8, 4(x2)
                SW      x1, 8(x2)
                ADDI    x10, x10, -1
                JAL     x1, loop1
                ADDI    x8, x17, 0
                LW      x10, 0(x2)
                ADDI    x10, x10, -2
                JAL     x1, loop2
                ADD     x17, x8, x17
                LW      x1, 8(x2)
                LW      x8, 4(x2)
                LW      x10, 0(x2)
                ADDI    x2, x2, 12
                JALR    x0, x1, 0
aucs-mbp:project aucuser$
```

```
main:
                AUIPC   x8,64528
                ADDI    x8, x8, 1081
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, 1014
                ecall
                ADDI    x17, x0, 8
                AUIPC   x10,64528
                ADDI    x10, x10, -28
                ADDI    x11, x0, 1024
                ecall
                ADD     x9, x0, x10
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, 972
                ecall
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, 986
                ecall
        loop1:  LB      x5, 0(x9)
                LB      x6, 1(x9)
                LB      x7, 2(x9)
                BEQ     x5, x0,  label1
                BEQ     x6, x0,  label1
                ANDI    x28, x5, 3
                SLLi    x28, x28, 4
                SRAI    x5, x5, 2
                ADD     x5, x5, x8
                LB      x5, 0(x5)
                ADDI    x17, x0, 11
                ADD     x10, x0, x5
                ecall
                ANDI    x29, x6, 15
                SLLi    x29, x29, 2
                SRAI    x6, x6, 4
                ADD     x6, x6, x28
                ADD     x6, x6, x8
                LB      x6, 0(x6)
                ADDI    x17, x0, 11
                ADD     x10, x0, x6
                ecall
                BEQ     x7, x0,  label1
                ANDI    x30, x7, 63
                SLLi    x30, x30, 0
                SRAI    x7, x7, 6
                ADD     x7, x7, x29
                ADD     x7, x7, x8
                LB      x7, 0(x7)
                ADDI    x17, x0, 11
                ADD     x10, x0, x7
                ecall
                ADD     x30, x30, x8
                LB      x30, 0(x30)
                ADDI    x17, x0, 11
                ADD     x10, x0, x30
                ecall
                ADDI    x9, x9, 3
                JAL     x0, loop1
label1:         ADDI    x17, x0, 10
                ecall
aucs-mbp:project aucuser$
```

```
aucs-mbp:project aucuser$ ./a.out sum_odd_bet_range/1.bin
main:
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, -4
                ecall
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, 44
                ecall
                ADDI    x17, x0, 5
                ecall
                ADD     x8, x0, x10
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, -15
                ecall
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, 0
                ecall
                ADDI    x17, x0, 5
                ecall
                ADD     x9, x0, x10
                ADDI    x10, x0, 0
                ADD     x18, x0, x8
                ADDI    x10, x0, 0
                ADD     x6, x0, x18
                ADDI    x17, x0, 4
                AUIPC   x10,64528
                ADDI    x10, x10, -42
                ecall
                ADDI    x10, x0, 0
                ANDI    x7, x6, 1
                ADDI    x28, x0, 1
                BEQ     x7, x28,  label1
                BEQ     x7, x0,  label2
label1:         ADDI    x18, x18, 0
                BEQ     x0, x0,  label3
label2:         ADDI    x18, x18, 1
label3: loop1:  BGE     x18, x9,  label4
                ADD     x10, x10, x18
                ADDI    x18, x18, 2
                JAL     x0, loop1
label4:         ADD     x29, x0, x9
                ANDI    x30, x29, 1
                ADDI    x31, x0, 1
                BEQ     x30, x31,  label5
                BEQ     x30, x0,  label6
label5:         ADD     x10, x10, x9
label6:         ADDI    x17, x0, 1
                ecall
                ADDI    x17, x0, 10
                ecall
aucs-mbp:project aucuser$
```