Wenzheng Kang
EN 605.204.83 SP21

EN605.204 – Computer Organization

Module 3 Assignment – MIPS Assembly Language

Question #1: Assembling MIPS Instructions

Assemble (MIPS -> binary/hex) the following instructions into their binary and hex equivalents (please show your work):

1. **add $t4, $t0, $t1**
2. **or $s0, $s2, $s3**
3. **andi $t3, $s0, 0x7123**
4. **jr $ra**
5. **j 0x1234**

$t0 $t1 $t4

① add instruction is an R-type instruction. The format is | opcode | rs | rt | rd | shamt | funct |

For add instruction, the opcode is $0_{10} = 000000_2$.
Refering to Green Sheet, we get the mapping of registers to integers,
Since $t0 = 8_{10} = 01000_2$, $t1 = 9_{10} = 01001_2$, $t4 = 12_{10} = 01100_2$
Shamt $= 0_{10} = 00000_2$. Refer to the Green sheet, we get function code $20_{16} = 100000_2$
Now we get the R-type function format, 000000 01000 01001 01100 00000 100000
and its hex equivalent 0 1 0 9 6 0 2 0

② Or instruction is R-type and format is the same as ①.
opcode $= 0_{10} = 000000_2$. Referring to Green sheet, $S0 = 16_{10} = 10000_2$, $S2 = 18_{10} = 10010_2$
$S3 = 19_{10} = 10011_2$. Shamt $= 0_{10} = 00000_2$. function code $25_{16} = 100101_2$.
Now we get the R-type format 000000 10010 10011 10000 00000 100101,
its hex equivalent 0 2 5 3 8 0 2 5

③ andi is I-type. format is | opcode | rs | rt | immediate |
opcode 001100_2. By the Green sheet, $t3 = 11_{10} = 01011_2$, $S0 = 16_{10} = 10000_2$,
$7123_{16} = 0111\ 0001\ 0010\ 0011$
Now we get I-type format 001100 10000 01011 0111 0001 0010 0011
its hex equivalent 3 2 0 B 7 1 2 3

④ jr is R-type. opcode $= 000000_2$. $ra = 31_{10} = 11111_2$ | 0 | 31 | | | | 8 |
binary 000000 11111 0000000000 00000 001000
hex: 03E 00008

⑤ j is J-type format. opcode $= 00010_2$. Target address = 1001 0001 1010 0001 1010 000000
Binary is
hex is

Question #2: Disassembling MIPS Instructions

Disassemble (binary/hex -> MIPS) the following instructions from binary/hex into their MIPS equivalent (please show your work)::

1. **00000010100101101000100000100010**
2. **0x01694024**
3. **00110101101101100001001000110100**
4. **0x2288FACE**

① opcode = $000000_2 = 0_{10}$ Then it's an R-type instruction.

opcode	rs	rt	rd	shamt	funct
000000	10100	10110	10001	00000	100010
$=0_{10}$	$=20_{10}$	$=22_{10}$	$=17_{10}$	$=0_{10}$	$=34_{10}$
	$=\$s4$	$=\$s6$	$=\$s1$.		$=22_{16}=$ Sub

Referring to Green sheet,

∴ Instruction is **Sub $s1, $s4, $s6**

② $01694024_{16} = 1011\ 0100\ 1010\ 0000\ 0001\ 0010\ 0100$.

opcode	rs	rt	rd	shamt	funct
000000	01011	01001	01010	00000	100100
$=0_{10}$	$=11_{10}$ $=\$t3$	$=9_{10}$	$=8_{10}=\$t0$.		$=36_{10}=24_{16}=$ add.

∴ Instruction is **add $t0, $t3, $t1**

③

opcode	rs	rt	Immediate
001101	01101	10110	0001001000110100

$=13_{10}=D_{hex}$ ⇒ I-type instruction.

$rs = 13_{10} = \$t5$, $rt = 22_{10} = \$s6$, immediate $= 4660_{10} = 1234_{16}$.

∴ Instruction is **ori $s6, $t5, 4660** = **ori $s6, $t5, 1234_{16}**

④ $2288FACE_{16} = 001000\ 10100\ 01000\ 11111\ 01011\ 001110$.

opcode	rs	rt	immediate
$=8_{10}=8_{16}=$ addi	$=20_{10}=\$s4$	$=8_{10}=\$t0$	$=64206_{10}=FACE_{16}$.

⇒ R-type instruction

addi $t0, $s4, 64206$_{10}$ = **addi $t0, $s4, $FACE_{16}$**

Question#3: Endianness

Assemble the following instructions from MIPS to hex and give the hex representation in both big- and little-endian format:

1. `or $s4, $t2, $t7`
2. `andi $t6, $s0, 0x4040`

or (little)	0X25	0XA0	0X4F	0X1
Memory Address	0x0	0x1	0x2	0x3

or (big)	0X1	0X4F	0XA0	0X25
Memory Address	0x0	0x1	0x2	0x3

andi (little)	0X40	0X40	0X0E	0X22
Memory Address	0x3	0x2	0x1	0x0

andi (big)	0X22	0X0E	0X40	0X40
Memory Address	0x3	0x2	0x1	0x0

① or $s4, $t2, $t7

000000_2 20_{10} 10_{10} 15_{10}.

$= 10100_2 = 01010_2 = 01111_2$.

Green sheet, function code for or.

$25_{16} = 100101_2$.

$000000\ 01010\ 01111\ 10100\ 00000\ 100101 = 14FA025_{16}$.

② andi $t6, $s0, 4040_{16}.

001000 14_{10} 16_{10} $= 010000\ 00010\ 00000_2$.

$= 01110_2 = 10000_2$

$001000\ 10000\ 01110\ 010000\ 00010\ 00000_2$.

$= 220E4040_{16}$.

Question 4: Registers

Please answer each of the following questions and justify your answer when needed.

1. **To which value is the $zero register hard-coded?**
2. **Which register would you use to return a value from a function call?**
3. **Which two registers are used for storing data on the call stack?**
4. **What is the difference between 's' and 't' registers?**
5. **Should the $at register be used by developers? Why/why not?**

1. $Zero register is hard-wired to the integer value 0.

2. Registers $V0 and $V1 are used for return value from a function call.

3. $SP, $fp are used for storing data on the call stack.
When the callee computes the return value, it places the value in $V0 or $V1.
Callee restores stack pointer and now the stack pointer points to the return value. $SP is reserved across a call.
If the callee uses a frame pointer, then it usually sets it to the stack pointer. The previous stack pointer must be saved on the stack before this happens.

4. t registers are temporary caller saved and can be used without saving their values. s registers are temporary callee saved and a called function must preserve their values.

5. $at is reserved for pseudo-instructions.
If a programmer find it hard to write code, then he can use pseudoinstructions. which adds clarity to the program by making the intention of the programmer more clear.
So programmers should use $at register.