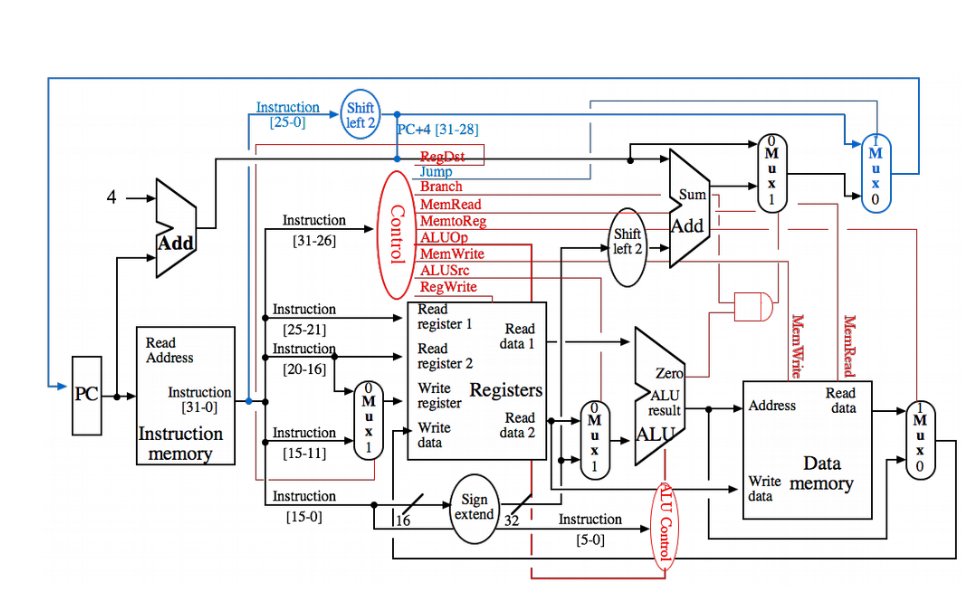
**Review Session Packet 4**

*(Make sure to review the questions on Homework 4 if you want more practice!)*

1. Trace out the pathway on the following CPU diagram in order to find the control bits for the given instructions. Then, fill in the control bit values on the provided table. If the instruction is not possible with the current version of our CPU, explain why.



| **Instruction** | **OR** | **SLTI** | **BNE** | **LB** |
| --- | --- | --- | --- | --- |
| *ALUsrc* | 0 | 1 | not possible | not possible |
| *aluOp* | 1 | 3 |  |  |
| *bInvert* | 0 | 1 |  |  |
| *Branch* | 0 | 0 |  |  |
| *Jump* | 0 | 0 |  |  |
| *MemWrite* | 0 | 0 |  |  |
| *MemRead* | 0 | 0 |  |  |
| *MemToReg* | 0 | 0 |  |  |
| *RegDst* | 1 | 0 |  |  |
| *RegWrite* | 1 | 1 |  |  |

2. Encode the following instructions into both binary and hexadecimal.

sra $t9, $s4, 3

special shift instruction, r-format.   
opcode = 0, rs = $s4, rt = 0, rd = $t9, shamt = 3, funct = 3

000000\_10100\_00000\_11001\_00011\_000011

sb $v0, 0xAF($a3)

i-format

opcode = 40, rs=$a3, rt=$v0, imm = 0xAF  
101000\_00111\_00010\_0000000010101111

3. Translate the following hexadecimal statement into its respective MIPS instruction.

0xAF31\_0400

1010 1111 0011 0001 0000 0100 0000 0000

opcode = 101011, rs = 11001, rt = 10001, imm = 0000010000000000

opcode = 43, rs = 25, rt = 17, imm = 1024

sw $s1, 1024($t9)

4. Suppose you have a MIPS instruction stored in a variable named inst. Write a snippet of

either Java or C code that extracts the rd field and stores it into another variable named d.

int d = (inst >> 11) & 0x1F

5. Now, do the same, but with MIPS code. Assume inst is stored in $s0 and you want to put rd

into $s1. Do not modify any other register.

sra $s1, $s0, 11

andi $s1, $s1, 0x1F

6. What’s the difference between the MIPS assembly instructions j, jal, and jr?

j = jump to an address

jal = jump and link to an address (the link part puts the return address in $ra)

jr = jumps to an address stored in a register, typically used with $ra

7. When do you put temporary variables on the stack? When do you put saved registers on the

stack? Why?

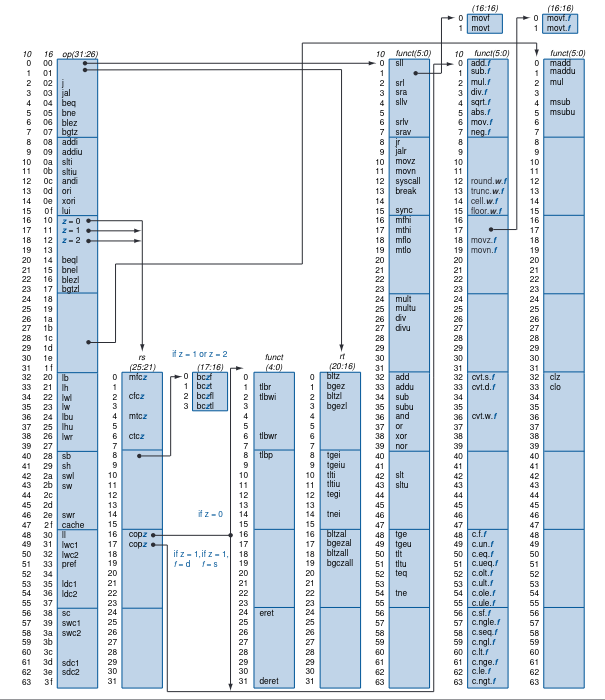
you put temp variables on the stack before calling a function and you put saved registers on the stack as the first operations of a function call.

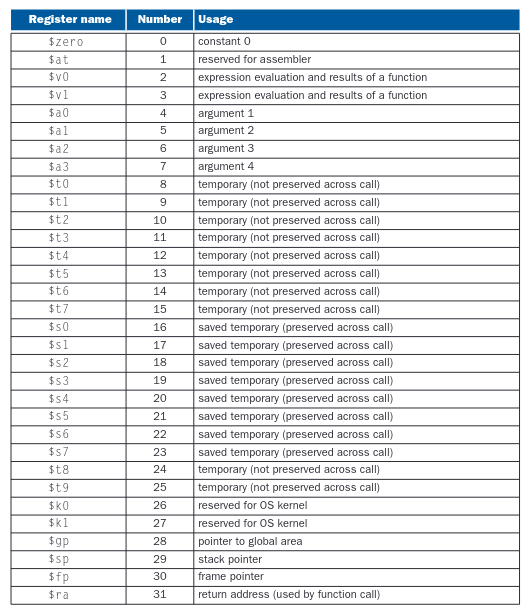
8. What’s the difference between absolute addressing and PC relative addressing? When

do you use one over the other?

Absolute addressing gives the entire address while PC relative address just gives an offset relative to the current program counter. j instructions use absolute addressing while branch instructions use relative addressing.

**References**

****

****

**Helpful Link:**

https://en.wikibooks.org/wiki/MIPS\_Assembly/Instruction\_Formats