

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Comp 120 Computer Organization
Spring 2005

Problem Set #6

Issued Thursday, 3/3/05; Due Thursday, 10/3/05

Homework Information: Some of the problems are probably too long to be done the night before the due date, so plan accordingly. Late homework will not be accepted. Feel free to get help from others, but the work you hand in should be your own.

Problem 1. “Compiler Appreciation”

Translate the following code fragments (written in C) into MIPS assembly language. Use the general approach to compilation shown in lecture (allocate variables in memory). You don't have to write an optimized assembly language unless you are into that sort of thing. Comment your assembly code to show the correspondence between C-language and assembly-language constructs. Just show the executable code—you can assume that the necessary storage allocation for each variable or array has already been done and that a label has been defined for each variable and the first entry of each array. Furthermore, you can assume that all variables have been allocated into the lower 32K bytes of memory, and that all variables and arrays are C integers, i.e., 32-bit values.

(A) `y = 4*x - 1;`

(B) `dst[row*32 + col] = src[row*32 + col];`

(C) `if (x > 0)`
 `abs = x;`
 `else`
 `abs = -x;`

(D) `for (j = 0; j < n; j += 1)`
 `a[j] = 2*j + 1;`

(E) `a[a[x]] = a[a[a[x]]];`

Problem 2. “Faking it”

MIPS assembly language provides opcode mnemonics for instructions that are not part of the instruction set architecture. For the most part, these pseudoinstructions can be generated using a sequence of one or more “true” MIPS instructions.

Find a “true-instruction” equivalent for each of the following pseudo-instructions (some are official MIPS pseudoinstructions, others are made up). Each of these can be implemented using only one real MIPS instruction. Discuss of your implementations, if any, and whether or not your implementation is unique (i.e. could some other instruction be used to achieve the same effect).

(A) `move rA, rB`
 $\text{Reg}[rA] \leftarrow \text{Reg}[rB]$
 Move register rB to rA

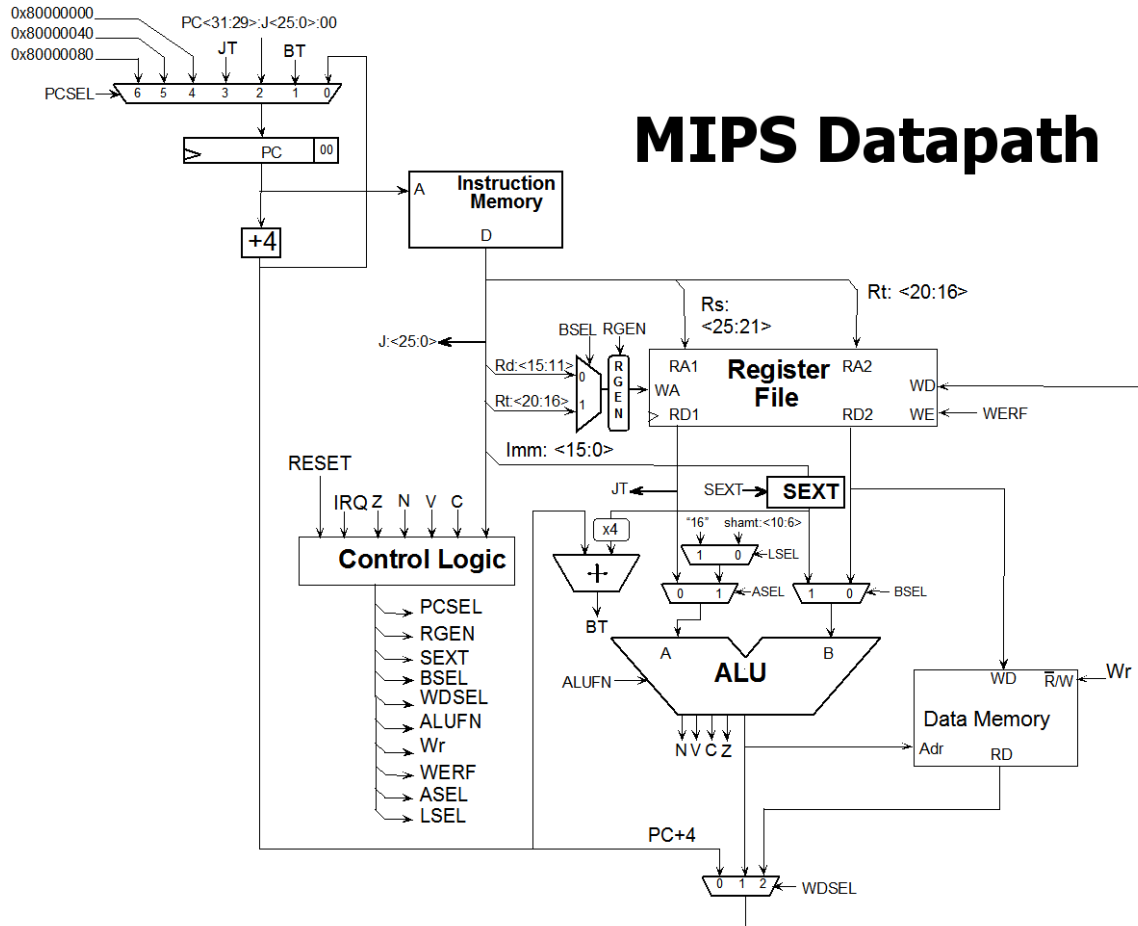
(B) `neg rA, rB`
 $\text{Reg}[rA] \leftarrow -\text{Reg}[rB]$
 Put the negative of register rB into register rA

(C) `not rA, rB`
 $\text{Reg}[rA] \leftarrow \sim\text{Reg}[rB]$
 Put the bitwise complement of register rs into register rd

(D) `inv rA, rB`
 if ($\text{Reg}[rB] == 0$)
 $\text{Reg}[rA] \leftarrow 1$
 else
 $\text{Reg}[rA] \leftarrow 0$
 Put the logical negation of register rs into register rd

(E) `slez rd, rs`
 if ($\text{Reg}[rB] <= 0$)
 $\text{Reg}[rA] \leftarrow 1$
 else
 $\text{Reg}[rA] \leftarrow 0$
 Set rd to 1 if rs is less than zero, otherwise set rd to 0

Problem 3. “Out of Control”



Fill in the missing entries of the Control Logic ROM, based on the data path shown above.

Opcode	PCSEL	RGEN	SEXT	BSEL	WDFSEL	ALUFN				Wr	WERF	ASEL	LSEL
						Sub	Bool	Shft	Math				
sub	0	0X	X	0	1	1	XX	0	1	0	1	0	X
sll	0	0X	X	0	1					0	1		
andi	0									0			
lw										0			
sw										1			
j						X	XX	X	X	0			
jal						X	XX	X	X	0			
lui										0			