Series 3

Solutions

Theoretical exercises

1. Performance calculations

Suppose a CPU is clocked at 500 Mhz. Suppose further that the CPU performs the following operations (with the specified amount of time):

ALU 4nsec, LOAD 8nsec, STORE 6nsec, Branch 6nsec.

You can assume that all operations are carried out equally often.

- How much faster/slower is a machine that needs 6 clock cycles for the LOAD instruction?
- How much faster is a CPU when its ALU works twice as fast?

Instruction	time	CPI	Q1	Q2
ALU	4 nsec	2		1
LOAD	8 nsec	4	6	
STORE	6 nsec	3		
BRANCH	6 nsec	3		
Mean		3	3.5	2.75

A1) $\frac{3.5 - 3}{3} \Rightarrow 14\%$ slower

A2) $\frac{3 - 2.75}{3} \Rightarrow 8.3\%$ faster

2. Using a stack in subroutines

Give two possible reasons why one needs the stack for assembler subroutines.

- Cache argument (if number of argument is >4)

  only four registers could be used for the arguments, if the number of argument is more than 4, then the argument should be saved on stack

- s-Register to save

  s-Register should not be changed in the subroutines. If any s-Register is used in the subroutine, to avoid to be overwritten, s-Register should be saved on stack

3. ALU & Most Significant Bit

Why does the ALU has to be built differently for the most significant bit and for the remaining bits?

- Overflow detection (add and subtraction)
- Support for slt

4. ALU & SLT (1)

- What happens during the slt command in the ALU?
- How does the ALU support the slt command?

- Control lines for Ainvert- and Binvert-Muxes to 01
- Control lines for Operation-Muxe to 11
- The sign bit serves as the Less signal for the 0^{th} box
- Get output from input value Less

	ALU Control lines		
Function	Ainvert	Binvert	Operation
and	0	0	00
or	0	0	01
add	0	0	10
subtract	0	1	10
slt	0	1	11
nor	1	1	00

4. ALU & SLT (2)

- What happens during the slt command in the ALU?
- How does the ALU support the slt command?

- A slt B = 000 … 001 if A<B, i.e, if A-B<0
- A slt B = 000 … 000 if A>=B, i.e. if A-B>=0

Thus, each 1-bit ALU should have an additional input (called "Less"), that will provide results for slt function. This input has value 0 for all but 1-bit ALU for the least significant bit.

For the least significant bit Less value should be sign of A-B

5. Pop and push

Specify how a pop and push can be implemented with the MIPS instruction set.

Push – You can push one or more registers, by setting the stack pointer to a smaller value and copying the registers to the stack

```
addi $sp,  $sp, -8 // reserve space
sw   $s1, 4($sp) // store rgister $s1
sw  $s0, 0($sp) //store register $s0
```

Pop - You can pop one or more gisters, by coping the data from the stack to the registers, then to add a value to the stack pointer

```
lw  $s1, t($sp)  // load register $s1
lw, $s0, 0($sp) // load register $s0
addi  $sp, $sp, 8 //free up space
```

6. loadi

Specify how the loading of a (32-bit) constant into a MIPS registers can be implemented with the MIPS instruction set.

Example

$$\underbrace{(0000\ 0000\ 0011\ 1101)}_{(61)_{10}}\underbrace{(0000\ 1001\ 0000\ 0000)}_{2304_{10}}$$

Load I

lui $s0, 61

ori $s0, $s0, 2304

0000 0000 0011 1101 0000 0000 0000 0000

0000 0000 0011 1101 0000 0000 0000 0000
0000 0000 0000 0000 0000 1001 0000 0000

7. ALU: OPCodes

Describe how the control bits of the ALU must be set for the following commands: and or add subtract slt nor

Explain in addition the relationship between these bits/commands and the individual elements of the ALU.

Example nor

nor a,b = not (a or b)

= not a and not b

AInvert = 1 BInvert = 1

Operation = 00

	ALU Control lines		
Function	Ainvert	Binvert	Operation
and	0	0	00
or	0	0	01
add	0	0	10
subtract	0	1	10
slt	0	1	11
nor	1	1	00

Programming exercises

main

```
int main ( int argc, char** argv ) {
    verbose = TRUE;
    if (argc != 3) {
        printf (”usage : %s expression filename\ n”, argv[0]);
        exit (EXIT FAILURE) ;
    }
    printf (”Input :   %s\n” , argv [1]) ;
    printf(”Postfix:  ”);
    compiler(argv[1], argv[2]);
    printf(”\nMIPS binary saved to %s\n”, argv [2]) ;
    return EXIT SUCCESS ;
}
```

loadFile

```
void loadFile(char* filename) {
    FILE *file = fopen(filename ,”r”);
    byte *buf = defaultMemoryData ;
    if(!file) {
        ERROR(”Can’t open file %s”, filename);
    }
     else {
        while (!feof(file)) {
            fread(buf, 1, 4, file) ;
            buf+=4;
        }
        fclose ( f i l e ) ;
    }
}
```

error

```
void error (const char *functionName , const char *fileName , int
      lineNumber , char * message, ...) {
        va_list details;
        printf(”%s in %s, line %i: ”, functionName , fileName , lineNumber ) ;
        va_start(details , message); // Initialize a variable argument list
        vprintf(message, details);
        printf (”\n”) ;
        va_end(details); //End using variable argument list
        exit (EXIT_FAILURE) ;
}
```