Ryan Young

CSE 3666

CSE 3666 Midterm #1 Study Guide

Basic MIPS:

MIPS is Big Endian meaning that it stores higher numbers first at the lower memory address. Aka it stores the higher values first, because memory goes from low address to high addresses.

Registers:

Temporary Registers: $t0 through $t9

Saved Variables: $s0 through $s7

Categories:

Arithmetic

Data Transfer:

Loading from memory to registers: lw $t0, 32($s3), offset(base-register)

Saving from register to memory: $t0, 32($s3), offset(base-register)

Integers stored in memory takes 4 bits so you if you want the index of 8 then you need to multiply it by 4. Chars are represented by one bit so they do not need to be multiplied by 4.

Loading small constants into variables: lui rt, constant, this will copy the 16-bit constant to left 16bits of rt and clears right 16 bits of rt to 0 aka it pads the right with 0’s.

Logical

Jump and Branch

Function or procedure

Numbers:

Binary: base 2

Octal: base 8

Decimal: base 10

Hexadecimal: base 16

Converting from base 10 to another base you divide by the base you want to go to and work your way down, save the remainder and the read from bottom to top. Just remember when you divide a number smaller than the base you are going to the remainder becomes 0.

Converting from binary to another base, group by the 2x = the base you want to go to. So, if the base is 8 then x is three, so you group the binary numbers in groups of 3. Convert each of these groups of three to an octal equivalent and then keep moving. If it was hexadecimal, then you would group in groups of fours.

To convert from octal or hexadecimal you just reverse the method.

Addition:  
Need to remember to carry when its 1+1 and make sure to check for overflow or underflow when talking about signed binary numbers.

Multiplication:

Easy everything is zero except 1 \* 1

Subtraction:

Straight forward unless it’s a negative number then you need to check for under flow.

2’s Complement:

Most significant bit is the signed bit a 1 is negative and a 0 is a positive number.

Positive numbers remain the same, if the number is negative you convert all 0’s to 1’s and 1’s to 0’s and then add 1 to the number. The most negative number possible would then be 1000 ….0000 And the most positive number would be 0111 …..1111. When you need to pad a signed number, you need to pad with the signed digit.

ASCII:

Uses 7 bits to represent 128 characters: which includes digits, English letters, and special characters plus 33 control characters. So, 65 is ‘A’ and 66 for ‘B’ etc… All ASCII can be stored in a byte.

Memory:

Main memory used for composite data: arrays, structures, and dynamic data. Each memory is an array of bytes, each byte is pointed by an address, each address identifies a byte, and a 32 bit address space supports 4 GiB memory. In order to perform operations on data you need to load it into registers first. MIPS is also Big Endian which means the most significant byte at least address of a word.

Converting MIPS to Binary Code:

R-Type Format ( the basic ones ):

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

OP code rs rt rd shamt funct

OP code: what type of operation to use

Rs: register source, this is the first register in an operation

Rt: register target, this is the second register in an operation

Rd: register destination, this is the register that an operation is going to be set equal to.

Shamt: Only used in shift operations this is how far to the left or right to shift

Funct: this decides which specialization of an operation to use ie whether to use add or subtract from the 0 operation.

I-Format Instructions ( immediate arithmetic and load/store operations ):

6 bits 5 bits 5 bits 16 bits

OP code rs rt constant or address

Rt is now the destination register, constant has to be between -215 and 215-1

Address: offset added to base address in rs, address = offset+base address

In save word and save load situations rs is the base register and rt is the destination register. Need to remember when adding the offset plus base register address you need to use sign extended notation aka pad with the sign bit not just 0’s. NEED TO DOUBLE CHECK HERE LOOKS LIKE IN EXAMPLE YOU ONLY NEED TO USE THE OFFSET HERE.

Advanced MIPS:

IF Statements:

Should be using the bne which is branch if not equal, so we if we can check if two values are not the same and then either branch to an exit function or branch to another function to do something else. So, this way if it is equal then it won’t branch, and it can keep moving down the lines in the program.

Looping:

Loop: Do something

Do some more things

Check a condition to exit

J Loop

So at the end it will jump back to the same function and run again, this being said you don’t have to link because you are staying in the same function. If you are jumping to another function and want to come back to this one you may need to use jal (jump and link).

When accessing indexes in arrays, make sure to shift left logical twice to multiple the I value by 4 so we can use it to index. Then add that shifted I value to the base address of the register. This will move us to the next address where we can then use load word and take the 0($register) of it because we already indexed!!!

Less than and Greater than:

Slt rd, rs, rt this will check if rs is less than rt and if it is set rd to 1 else rd = 0. Slti will do the same thing but check if a register’s value is less than an immediate. You can check if something is less than or greater than and then branch by setting the value to a temp variable and then checking that to determine if we should branch. It is better to do this because it is faster to check for equal or not equal rather than checking for greater than or less than. You can also use sltu and sltui if you want to compare unsigned integers.

JAL:

Jal func # will jump to func and store this line in $ra

When jr $ra is called it will now resume from the line below the one stored in $ra

If you are already at the highest level meaning you are in your main function, then jr $ra will return (just like in C when you use a return statement).

Encoding and Decoding:

Jump: you use the full address because we have enough bits. To calculate the 26-bit number you take off the last 2 bits on the right and then remove the first four bits. This is because it has to be a multiple of 4 and the other first four its will get added later from the pc register.

Branch: You need to add the immediate to the address that you came from in order to find the target address. This can be calculated by multiplying the number of lines by 4.

Remember going back up you have to use a negative number and have to remember to use two’s complements when encoding negative numbers. Remember you start after the line that you branch from because you have already processed that line, when counting the number of lines to find the differential.