MITOCW | MIT6_004S17_10-02-08-01_300k

As presented in lecture, in this course, we use a simple 32-bit processor called the Beta.

The Beta works on 32-bit instruction and data words.

However, the addresses in memory are specified in bytes.

A byte is made up of 8 bits, so each 32-bit instruction consists of 4 bytes.

That means that if you have two instructions A and B in consecutive memory locations, if A is at address 0x100, then B is at address 0x104.

Now, suppose that you are given the following piece of code.

The . = 0 notation tells you that your program begins at address 0.

You can assume that execution begins at this location 0 and halts when the HALT() instruction is about to be executed.

We want to determine what value ends up in R0 after this instruction sequence has been executed.

Note that we are working with hexadecimal numbers in this code and we want our answer to also be in hexadecimal.

This code begins with a LD operation into register R0.

The load, uses the value of R31 + c as the source address for the load.

Since R31 = 0, this means that the value stored at address c is being loaded into R0.

So after the LD, R0 = 0x300.

Next an ADDC of R0 with the constant b is performed and that result is stored back into R0.

The .=0x200 notation immediately preceding the "a" label, tells us that address a = 0x200.

This means that address b = 0x204, and c = 0x208.

So if we are adding the constant b to R0, R0 now becomes 0x300 + 0x204 = 0x504.

Now lets take a look at this short piece of code.

Our goal is to determine the value left in R0 in hexadecimal.

The . = 0 notation once again tells us that our first instruction (the branch) is at address 0.

The branch instruction then branches to location . + 4 = 0 + 4 = 4.

This is the address of the HALT() instruction.

In addition to branching to the HALT() instruction, a branch instruction also stores the address of the instruction immediately following it into the destination register, R0 in this case.

The address of the next instruction is 4, so R0 = 0x4.

Let's take a look at what this code is doing.

It first loads the contents of address x into R0, so R0 = 0x0FACE0FF or 0xFACEOFF for short.

It then moves the constant 0 into R1, so R1 = 0.

It now enters the loop where the ANDC puts into R3 the least significant bit of R0.

The ADD increments R1 if R3 equals 1.

This means that if the least significant bit of R0 was a 1, then R1 is incremented by 1, otherwise R1 stays the same.

The shift right constant then shifts R0 to the right by 1.

This makes R0 have a 0 in the most significant bit, and the top 31 bits, of what R0 used to be, are shifted over by one position to the right.

Note that this means that the least significant bit of the old R0 is now completely gone.

That's okay though because we already incremented R1 based on that original least significant bit of R0.

The BNE, or branch on not equal, then branches back to loop as long as R0 is not equal to 0.

This means that what this loop is doing is looking at the current least significant bit of R0, incrementing R1 if that bit is 1, and then shifting that bit out until all bits have been shifted out.

In other words, it's counting the total number of ones in the original value loaded from address x.

The loop ends when all the 1's have been counted at which point R0 is left with a 0 in it because all the 1's have

been shifted out.

R1 is left with the number of 1's in the data 0x0FACE0FF equals in binary 0000 1111 1010 1100 1110 0000 1111 1111.

There are 19 ones in 0x0FACE0FF, so R1 = 19 = 16 + 3 which in hexadecimal = 0x13.

In this piece of code, the CMOVE first sets the stack pointer to 0x1000.

Then a PUSH(SP) operation is performed.

Lets first understand what a PUSH instruction does.

A PUSH instruction is actually a macro made up of two beta instructions.

To push a value onto the stack, the stack pointer is first incremented by 4 in order to point to the next empty location on the stack.

This sets SP = 0x1004.

Then, the contents of register Ra, which is being pushed onto the stack, are stored at the memory location whose address is SP-4 which is address 0x1000.

Now looking at the actual PUSH operation performed here, we are performing a PUSH of stack pointer so the Ra register is also the stack pointer.

This means that the value stored at location 0x1000 is actually the value of SP which is 0x1004.

So the value that got pushed onto the stack is 0x1004.