Matthew Michaelis, CSCI 5593

February 5th, 2019

**Homework Assignment 1**

**1:**

My website can be found at: <https://matt-mich.github.io/CSCI5593/>

**2a:**

load R1 , A

load R2 , B

add R3 , R1 , R2

store C , R3

**2b:**

This is very similar to a typical RISC architecture.

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | Destination  register | Source  register 1 | Source register 2 |
| 6 bits | 5 bits | 5 bits | 5 bits |

Since there are 32 possible arithmetic opcodes with additional codes for load/store operations (presumably less than 32), more than 32 opcodes must exist, thus at least 6 bits are needed to identify which one is being called. Additionally, the destination and the input parameters are all registers, so each one requires an equivalent number of bits. There are 32 registers, so we again need 5 bits to address each one. We won’t be using the full 32 bits in this format, only 20, so there will be 12 bits left blank.

**2c.**

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | Destination  register | Base  register | Displacement  (signed) |
| 6 bits | 5 bits | 5 bits | 16 bits |

This is fairly similar to the previous instruction format up until the second referenced register. Instead of acting as a source of data, the base register stores a base memory address (32 bits), and the remaining 16 bits are used as a displacement from that address. We choose 16 bits as that is all the remaining bits in the opcode, and the displacement would ideally be as big as possible (with an upper limit of 32 bits, since there are only 2^32 memory locations).

**3a.**

ld\_to\_Ac A

add B

store\_from\_Ac C

**3b.**

|  |  |  |
| --- | --- | --- |
| Addressing mode | opcode | Immediate value/Memory Address |
| 1 bit | 7 bits | 24 bits |

This is a slightly different format from the RISC-like instruction formats from before, as this system is more oriented towards specific use registers than GPRs. Because of this, the opcodes will be geared towards loading to a specific register. For instance, it would have a load command specifically to load into the accumulator, rather than a generic load that takes two parameters. Additionally, we require a single bit to determine the addressing mode, as specified in the prompt. If the bit is set, the memory address will be added to whatever is in register X, and if it’s reset, it will be used as a standalone address. Because of the fact that registers are identified by operation code, only one field is necessary as a memory address or immediate value field (Usage depends on the opcode), and for that reason we make it as large as possible. Since we need 1 bit for addressing, 7 bits for the opcodes, we have 24 bits left over for the memory address/value field.

**4a.**

|  |  |  |
| --- | --- | --- |
| Instruction class | Number of memory reads per instruction | Percent frequency in spice |
| Arithmetic | 1 | 50% |
| Data transfer | 2 (1 for instruct, 1 for operand load/store) | 41% |
| Conditional branch | 1 | 8% |
| Jump | 1 | 1% |

Let’s say there are 100 operations done, using the given percentages. That would mean 50 memory reads from arithmetic instructions, 82 reads/writes from data transfer, 8 from conditional, and 1 from jump. Now, we have a total of 50 + 82 + 8 + 1 = 141, with 41 of those being data accesses.

Thus the total percentage of memory accesses being data related is:

(41/141)\*100 = 29.078%

**4b.**

Now, assuming two-thirds of the data transfers are loads, and knowing that all instruction reading are reads, we can determine the percentage of memory accesses that are reads by subtracting the number of writes, or simply multiplying the number of data accesses

by one-third divided by the total number of accesses times 100 subtracted from 100 to get our final percentage:

100-((41\*(1/3))/141)\*100 = 90.307%