CSC 252
Written Assignment 1
8th October 2020
Due 15th October 2020

1. (20 points) Consider the following different styles of instruction set architectures:

   (a) Stack: All operations occur on the top of a stack. Only `push` and `pop` instructions access memory. All other instructions remove their operands from the stack and replace them with the result (the x86 floating point unit uses a version of this architecture). The number of stack entries that are retained on-chip is 2 (top two entries) — accesses to other stack positions are memory references. Additionally, the `dup` instruction duplicates the entry at the top of the stack.

   (b) Accumulator: A single architecturally visible register implicitly identified in most instructions. This type of architecture was used in the early days of computing when hardware (real-estate/area) was precious. In addition to register space, it also reduced code space since there was no need to specify the register operand.

   (c) Load-store: All operations occur in registers, and register-to-register instructions have three operands per instruction. There are 16 general-purpose registers, and register specifiers are 4 bits long.

   (d) Memory-to-memory: All three operands of each instruction are in memory.

   All architectures support the following operations: `add`, `sub`, and `neg`. `neg` negates the top of the stack, the implicit accumulator, the explicitly specified register, or the explicitly specified memory address, depending on the instruction set architecture style.

   The following shows the assembly code for the C statement `A = B + C;` (with comments separated by a #); note that the order of operands remains the same for operations where the operand order matters, e.g., `A = B - C`.

   - Stack:

     ```
     push AddressC   # Top=Top+4; Stack[Top] = Memory[AddressC]
     push AddressB   # Top=Top+4; Stack[Top] = Memory[AddressB]
     add             # Stack[Top-4]=Stack[Top]+Stack[Top-4]; Top=Top-4;
     pop AddressA    # Memory[AddressA] = Stack[Top]; Top=Top-4;
     ```

   - Accumulator (Acc):

     ```
     load AddressB    # Acc = Memory[AddressB]; Acc = B
     add  AddressC    # Acc = Acc + Memory[AddressC] or Acc = Acc + C; Acc = B + C
     store AddressA   # Memory[AddressA] = Acc; A = B + C
     ```

   - Load-store:

     ```
     load AddressB, regB      # regB = Memory[AddressB]
     load AddressC, regC      # regC = Memory[AddressC]
     add regB, regC, regA     # regA = regB + regC
     store AddressA, regA     # Memory[AddressA] = regA
     ```

   - Memory-to-memory:

     ```
     add AddressB, AddressC, AddressA    # A = B + C
     ```

   You may make the following assumptions about all four instruction sets:

   - The opcode is always 1 byte (8 bits).
   - All memory addresses are 2 bytes (16 bits).

- All data operands are 4 bytes (32 bits).
- All instructions are an integral number of bytes in length.
- For operand specifiers that are not an integral number of bytes, multiple specifiers within a single instruction may be packed into a single byte.

Byte Transfer = Instruction Bytes + Data Bytes

For example, a register load will require four instruction bytes (one for the opcode, one for the register destination, and two for a memory address) to be fetched from memory along with four data bytes. A memory-to-memory add instruction will require seven instruction bytes (one for the opcode and two for each of the three memory addresses) to be fetched from memory and will result in 12 data bytes being transferred (eight from memory to the processor and four from the processor to memory).

For the following C code, write an equivalent assembly language program in each architectural style (assume all variables are initially in memory and tno compiler or hand optimizations are performed):

```
a = b + c;
b = a + c;
d = a - b;
```

For the code sequence in each instruction style, calculate the instruction bytes fetched and the total number of bytes (instruction AND data) transferred (read or written) to or from memory. Which architecture is most efficient as measured by the code size? Which architecture is most efficient as measured by the total number of bytes trasferred to or from memory (the total number of instruction AND data bytes)? If the answers are not the same, why are they different?

2. (20 points) Consider a fictitious 16-bit computer with registers r0–r7. Memory is word-addressable, i.e., each address refers to a 16-bit quantity (as opposed to the traditional byte-addressable memory where each byte is addressable). Instruction operands may be addressed in any of the following eight addressing modes:

Syntax	Mode	Effect
r_n	Register	r_n
(r_n)	Register indirect	$M[r_n]$
$d(r_n)$	Displacement	$M[d + r_n]$
$@d(r_n)$	Displacement indirect	$M[M[d + r_n]]$
$(r_n)+$	Autoincrement	$M[r_n]; r_n \leftarrow r_n + 1$
$@(r_n)+$	Autoincrement indirect	$M[M[r_n]]; r_n \leftarrow r_n + 1$
$-(r_n)$	Autodecrement	$r_n \leftarrow r_n - 1; M[r_n]$
$@-(r_n)$	Autodecrement indirect	$r_n \leftarrow r_n - 1; M[M[r_n]]$

   (a) How many bits of an instruction are needed to specify an operand? (do not count the d displacement operands.)

   (b) r7 is also the program counter (PC), which during execution of an instruction points to the next instruction in sequence (assuming no change in control flow). We may therefore derive new modes by using r7 in one of the above modes. Give the appropriate addressing mode (using r7; just the addressing mode will suffice - no need for further details) for each of the following:

      (i) Immediate — Operand is the next word (after the currently executing instruction)
      (ii) Absolute — Operand is addressed by the next word
      (iii) PC-Relative — Operand is at PC + d
      (iv) PC-Relative-Indirect — Operand is addressed by the word at PC + d

   (c) In each of the above derived addressing modes, what must be done to take care not to execute operands?

   (d) Consider the following local declarations (in C). Local variables are addressed in C using a displacement from the frame pointer (r5 in this case):

```
int i, *pi, **ppi;
```

Integers are assumed to occupy one word (remember that only whole words may be addressed, so that the address of the next consecutive word may be obtained by adding or subtracting 1 from the current address). Also keep in mind that the content of pi and ppi is determined only during execution, i.e., `pi` may not point to `i` and `ppi` may not point to `pi`.

Provide the appropriate addressing mode for each operand (for those that can be directly addressed by an instruction on this hypothetical computer through a single addressing mode) under each of the following assumptions. You need not give the exact operand, the correct addressing mode will suffice.

(i) Nothing is in a register (except for the frame pointer in r5)

4	i	*pi	*(pi+1)	**ppi	**(ppi+1)	*(*ppi+1)
_____	_____	_____	_____	_____	_____	_____

(ii) i, pi, and ppi are in registers (in addition to the frame pointer in r5)

4	i	*pi	*(pi+1)	**ppi	**(ppi+1)	*(*ppi+1)
_____	_____	_____	_____	_____	_____	_____

3. (20 points) You are the lead designer of a new processor. The processor design and compiler are complete. However, your manager has requested that you spend an additional 6 months to improve your system. The base system has a clock rate of 1 GHz and the following measurements have been made of the expected workload:

Instruction class	CPI	Frequency
Integer	2	40%
Control flow	3	25%
Floating point	3	25%
Memory	5	10%

The hardware team claims that in 6 months it can improve the processor design to give it a clock rate of 1.2GHz, but with the following measurements for the same workload:

Instruction class	CPI	Frequency
Integer	2	40%
Control flow	4	25%
Floating point	3	25%
Memory	6	10%

The compiler team claims that in 6 months it will be able to improve the compiler by reducing the number of instructions of each type required relative to the base compiler.

Instruction class	Percentage of instructions executed vs. base machine
Integer	90%
Control flow	90%
Floating point	85%
Memory	95%

You are given the choice of employing only one of these teams. Which would you choose? Your objective is to minimize the execution time of the workload, which has a total of 100 million instructions on the base machine. Explain your answer and back it up quantitatively.