CS 4380 Projects (3 weeks)

Any instruction can be preceded by a label.
Random example of assembly code instructions:

```
COUNT   .INT   1
INDEX   .INT   0
NEXT    .INT   5
R       .BYT 'R'
A       .BYT 'A'
Y       .BYT 'Y'
START   LDR R0, COUNT
        LDR R1, INDEX
        ADD R1, R0
        STR R1, INDEX
        TRP 1
BEGIN   MOV R3, R0
        CMP R3, R5
        BNZ R3, BADX
        TRP 3
        TRP 0
```

Jump Instructions

Op Code	*Description*	*Operands*	*Value*
JMP	Branch to Label	Label	1
JMR	Branch to address in source register	RS	2
BNZ	Branch to Label if source register is not zero	RS, Label	3
BGT	Branch to Label if source register is greater than zero	RS, Label	4
BLT	Branch to Label if source register is less than zero	RS, Label	5
BRZ	Branch to Label if source register is zero	RS, Label	6

Move Instructions

Op Code	*Description*	*Operands*	*Value*
MOV	Move data from source register to destination register	RD, RS	7
LDA	Load the Address of the label into the RD register. This instruction should ONLY work if the label is associated with a DIRECTIVE. THIS command must NOT be used to get the address of an instruction.	RD, Label	8
STR	Store data into Mem from source register	RS, Label	9
LDR	Load destination register with data from Mem	RD, Label	10
STB	Store byte into Mem from source register	RS, Label	11
LDB	Load destination register with byte from Mem	RD, Label	12

Arithmetic Instructions

Op Code	***Description***	***Operands***	***Value***
ADD	Add source register to destination register, result in destination register	RD, RS	13
ADI	Add immediate data to destination register.	RD, IMM	14
SUB	Subtract source register from destination register, result in destination register	RD, RS	15
MUL	Multiply source register by destination register, result in destination register	RD, RS	16
DIV	Divide destination register by source register, result in destination register	RD, RS	17

Logical Instructions

Op Code	***Description***	***Operands***	***Value***
AND	Perform a boolean AND operation, result in destination register. This is a logical AND not a bitwise AND.	RD, RS	18
OR	Perform a boolean OR operation, result in destination register. This is a logical OR not a bitwise OR.	RD, RS	19

Compare Instructions

Op Code	***Description***	***Operands***	***Value***
CMP	Set destination register to zero if destination is equal to source; Set destination register to greater than zero if destination is greater than source; Set destination register to less than zero if destination is less than source.	RD, RS	20

Traps

Op Code	*Description*	*Operands*	*Value*
TRP	Execute an I/O trap routine (a type of operating system or library routine) using register R3. IMM Values 1, write integer to standard out 2, read an integer from standard in 3, write single character to standard out 4, read a single character from standard in Read or write a value from register R3.	IMM	21
TRP	Execute STOP trap routine. 0, stop program	IMM	21
TRP	DEBUG (OPTIONAL) IMM Value 99 If you use the TRP its output must be suppressed in the version of code you supply to me!	IMM	21

Directives

Directive	*Description*

.INT value	`Allocate space for an integer.`
	`Example:`
	`MONTH  .INT  12`
	`DAY    .INT  9`
	`YEAR   .INT  2005`
	`STUFF  .INT  9`
	`       .INT 17`
	`       .INT 42`
	`       .INT 53`

.ALN	`Align the next byte on a word boundary.  (NOT USED)`

.BYT value	`Allocate space for one byte.`
	`Example:`
	`N        .BYT  78  # Use the ascii code`
	`A        .BYT  65`
	`M        .BYT  77`
	`E        .BYT  69`
	`LETTER_A .BYT 'A' # Or use the character`
	`SCHOOL  .BYT 'U'`
	`        .BYT 'V'`
	`        .BYT 'U'`

Registers

Register	*Description*	*Value*
R[0…7]	General purpose integer registers named R0 through R7	0, 1, 2, 3, 4, 5, 6, 7
PC	Program Counter, can't move a value into this register from a MOV instruction but you can copy its value to another register.	8

CS 4380 Project 1

Implement a Virtual Machine, which can execute the Test Program outlined below.

Use the following outline for submitting all projects.
Submit your project via canvas.

Your project MUST be packaged in a zip file the name of this .zip is:
YourName_p1.zip

Where YourName is your name
p1 is the name of the project
Note: Canvas will allow you to resubmit your project as many times as you want. Only the most current version will be kept my Canvas and graded by me. I highly encourage you to submit early and often. If you wait to the last moment and Canvas, the internet or your computer doesn't work then THIS IS A MISTAKE on your part.

Your project must include:

1. Source Code for your project.

2. Notes put directly in Canvas that tells me **how to execute your program** and other facts you want me to know. Only needed if your project doesn't work as I would expect (vm,exe proj1.asm). If you know you have a bug tell me. I think it is better to know you have a bug than it is to look like I found a bug you have no knowledge of.

3. proj1.asm this is your assembly file. This is what I grade. This file must be accepted as a command line argument. I may rename the file just to check that your code really works. **Don't misname your project**

4. vm.exe this is your compiled executable. You must send an executable if your language doesn't support this then YOU Must ***build me a script to execute your program*** (vm.bat). Read if you don't understand what this means.

    - Your executable must accept a command-line argument of the assembly file that it uses.
        - vm.exe proj1.asm **Don't misname your project**
        - vm.bat proj1.asm **Don't misname your project**
            - As long as you leave me explicit instruction on how to run your scripting language in Canvas I will accept it but you really should learn to use the vm.bat file

    - Be very careful using **Visual Studio** to make your program every year students send me executes that will not execute on Windows because they are compiled incorrectly. Always execute your program outside of the development environment before sending it to me (or anyone else).
    - Your program **must not hang** after running. I will execute it at command-line and will expect it to finish when the program is done.

Important Notes:

- Be sure to suppress all debug output in the final asm/vm version you submit.
- Make sure you give me an executable that runs on Windows. You can bring an executable to my office to run.
  - THIS DOESN'T NEED TO BE YOUR PROJECT
  - NO, I WILL PRE-GRADE IT THEN LET YOU FIX IT
- Read the assembly file name from the command-line don't hard code it.
- Your vm must NOT hang at the command-line waiting for input after the program has completed!
- You are making an **awful assumption** if you think you can get a passing grade on a project without completing every element of the project.
  - I don't grade source code that doesn't compile and execute
  - I don't give you credit for programs that crash when run
  - This class is NOT like peewee soccer you will not be given credit for your effort only your achievements.
- Don't procrastinate this project can be completed by even an average CS student as long as you start today!
  - When Canvas closes it will not be reopened.
  - Don't send late projects to me via email. Complete them on time.
  - Submit Early… Submit Often…
- If you have questions about if the language you are using will execute on my machine come by my office and try it out.
  - NO I won't pre-grade your project
  - Remember you only need an example program not a completed project.

You <u>may not</u> change the syntax of the instructions! I will not grade it
You <u>may not</u> change the semantics of the instructions! I will not grade it
You <u>may not</u> add instructions. I will not grade it
When in doubt ASK!

Test Program

Write the following assembly program using your assembly language.
Place the following list of integers in memory
A = (1, 2, 3, 4, 5, 6)
B = (300, 150, 50, 20, 10, 5)
C = (500, 2, 5, 10)
Place your full name “Last Name, First Name” in contiguous memory.

Program Output (follow the output formatting exactly. I’m very picky on <u>***this***</u> assignment not so much on the others.)

1) Print your name “Last Name, First Name” on the screen. You don’t need the quotes but you do need the comma.
2) Print a blank line.
3) Add all the elements of list B together; print each result (intermediate and final) on screen. Put 2 spaces between each result. ← pay attention to this (e.g., `450`) ← pay attention to this
4) Print a blank line. ← pay attention to this
5) Multiply all the elements of list A together; print each result (intermediate and final) on screen. Put 2 spaces between each result. (e.g., `2`) ← pay attention to this
6) Print a blank line.
7) Divide the final result from part 3, by each element in list B (the results are <u>not cumulative</u>). Put 2 spaces between each result. (e.g., `  1`)
8) Print a blank line.
9) Subtract from the final result of part 5 each element of list C (the results are <u>not cumulative</u>). Put 2 spaces between each result. (e.g., `220`)

Create a directive for each element of the list and letter of your name.
Example.

```
A1      .INT    1
A2      .INT    2
A3      .INT    3
C       .BYT    'C'
u       .BYT    'u'
r       .BYT    'r'
```

Grading

The project is worth 100 points Systematic Deductions are taken off the top before individual Project Deductions are made. Systematic Deductions are like penalties for doing something a senior shouldn't do because they are just too fundamental and simple.

A basic outline of the grading process but this doesn't necessarily cover every case:

All Project grades in CS 4380 are Performance-based (Mastery Grades) meaning you will only receive credit for elements you complete correctly not things you attempt. You should never expect to receive credit for elements you work on but do nothing.

Systematic Deductions

Late (no excuses; submit early; submit often)	-100
Naming executable or asm file incorrectly vm.exe/vm.bat, proj1.asm	-10
Sending an un-executable project Doubles with each offense (most often a VisualStudio issue)	-20
VM hangs when finished	-10
VM crashes during execution	-50 or greater
Debug Output	-10
Failure to send all the elements in your zip	-10
Not following instructions which forces me to regrade your project (e.g., altering instructions)	-20

Project Deductions

Each Major element	-20 max

- Your Name, Add, Multiply, Divide, Subtract, Stop, Modify and Re-run

Projects with a grade of 60 or lower are rare but indicate significant issues in your development skills or time management. While it's still possible to pass the class with such a grade on a single project; such a grade generally indicates without a massive effort on your part, you'll end up not passing this class as you'll continue to have problems on future projects as well.

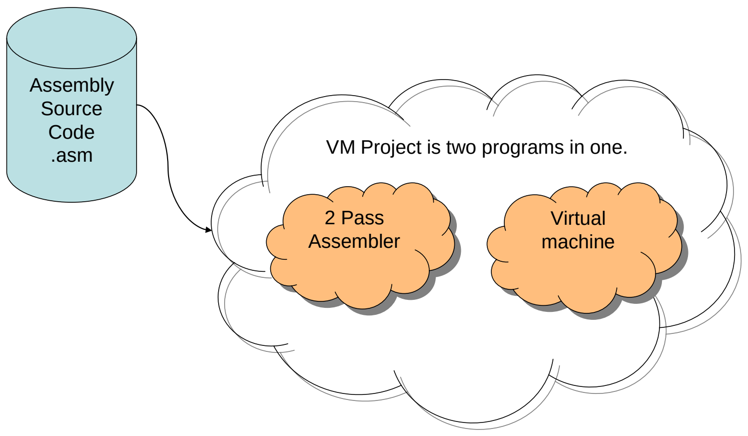
The issue isn't making one bad mistake and then having to cover for it the rest of the class by making good grades to pass. If you will sit down with Excel or a calculator you can easily see what I mean. The issue is most students who get a poor project grade will have poor grades on subsequent projects and marginal test grades.

Late projects will not be graded. Don't ask. However, if you take the time to bring it to my office and sit down with me, I will talk to you about it. Even a 0 on a single project doesn't mean you will fail the class if your test, homework and other project scores are high.

Hints:

Write the **simplest Assembly program** you need to complete the test program. **You don't need LOOPS or REGISTER INDIRECT ADDRESSING** to complete the project. Then write an assembler and a Virtual Machine to execute your program.

Do this as an incremental Development Project not one big problem.



Implement a **Two-Pass Assembler**. The instructions you implement are to be taken from the move instructions, arithmetic instructions, trap instructions. **Not all instructions are needed** for this project, but you will need to implement all these instruction (plus a few more) by the time you turn in the last project.

In the **first pass create a Symbol Table** (Associative Map)
The key of the Symbol Table is a Label (Code, Data) from an assembly program. The value associated with the key is the location of the Label (**Label → Address**). Compute the address of the Label during the first pass.

Instructions count for N bytes of space
.**BYT** Variables count for 1 bytes of space
.**INT** Variables count for K (a min of 4) bytes of space

Pass #1

Read a line of text
Break text on spaces
Find optional label
Find Operator—Find Operands—Increment counter by instruction
OR
Find Directive—Find Data—increment counter by data size
Generate Error Message
Load label to Symbol Table

👍👍 In the **second pass create byte code** for your program. Using the Symbol Table convert Labels to addresses. Your **byte code must be all numeric data**: (e.g., ADD is 13, R7 is 7, Label A is 1024, etc.)

Pass #2

Read a line of text
Break text on spaces
Find optional label
Find Operator—Find Operands—Lookup address for labels—Convert instruction to bytecode—Load bytecode to memory—Increment counter by instruction
OR
Find Directive—Find Data—Convert data to binary—Load binary data to memory—increment counter by data size
Generate Error Message

To parse instruction in the first and second pass read one (1) line of data at a time from your assembly file.

👍 Write a **Virtual Machine (VM)** that can execute your byte code. To make things simpler embed your assembler in your VM. Thus your program will work as:

- **Assembler Pass 1 -** Parse instruction & Load Symbol Table
- **Assembler Pass 2 -** Parse instruction, Create Byte & Load Byte Code to Memory
- **Execute Byte code Program**

Don't try to skip these steps. It will not work and it's not easier!
Separating your Assembler and VM will make completing CS4380 more difficult!! Don't do it!

👍 You will need the following address modes for this project

- **Direct** EA = Address
- **Register** EA = Register
- **Immediate** - may also be very useful

👍Memory layout is one of your biggest issues. There are a few major options to consider.
IF you have done Project 0 you are finished with this part of the project!

👍👍 **Traps 1 and 3 are used for output to the screen**. Note there is **NO TRAP THAT CAN WRITE OR READ A STRING**! Don't create one!

👍👍👍👍 **Strongly consider supporting comments**. Just have your assembler discard the comments when they are encountered. Comments will help me grade your projects, even more they will help you write your projects.

```
MUL     R5, R6          ; Multiply  A * B
```

👍👍👍👍 **Don't overlook TESTING.** I've already talked about using **TDD** along with incremental design, implementation and testing. There are two additional steps you can take to testing that will ensure the quality of your project and the products you built.

1. ***Don't approach testing as a process to prove your system works always approach testing a process to prove your system doesn't work. Don't run test you know will work runs tests you believe will fail!***
   a. If you believe the Earth is flat or the Sun circles the Earth or Apple invented the mouse or Java invented Garbage Collection you can find some evident to support this. However, it is simple to disprove each of these statements.
2. **Find two other students taking CS4380 and *once you have finish your projects compare the output of your projects*. It is virtually impossible for three of you**

get the same wrong solution UNLESS you're working too closely in developing your solution. CS4380 is NOT A GROUP Project!

a. Don't share code THIS IS CHEATING.
b. Swapping ideas is OKAY!

How to approach a big project (School or Work):

- Never build something because it is cool or cleaver.
- Only build what is needed to complete the project.
  - If you finish early you might have free time!
- Focus on what you must accomplish not what you want to do.
- Consider the order in which you must build things to make the best progress.
- Use both top-down and bottom-up design & implementation.

Assembler Pass 1

Read Assembly Source File

Group chars into Tokens (Lexical Analysis)

Regular Expressions
Tokenizer (C strtok)
String Compare
Finite State Machine

Check Syntax of Instructions (Syntax Analysis)

Ensure that all instructions are part of the defined Assembly Language.
[optional]

Directives:

[Label]	.INT	Integer
	.ALN	
[Label]	.BYT	Integer

Instructions:

[Label] Operator Operand1 [Operand2]

Operand1 is Register, Label or Immediate
Operand2 is Register, Label or Immediate

Load Defined Labels into a Symbol Table

Labels on Directives (Memory allocation)
Labels on Instructions (Branch locations)
And resolve there address!

Symbol Table
Label $\rightarrow$ Memory Address + other
Check that Defined Labels are unique

Assembler Pass 2

Check that Referenced Labels are defined in the Symbol Table

LDR	R1, NUTMEG	is label NUTMEG defined (what is its address?)
JMP	NEXT	is label NEXT defined (what is its address?)

Generate Code (Byte Code)

We have a very simple LOADER on this project when you moved bytecode and data to memory you're implementing a LOADER.

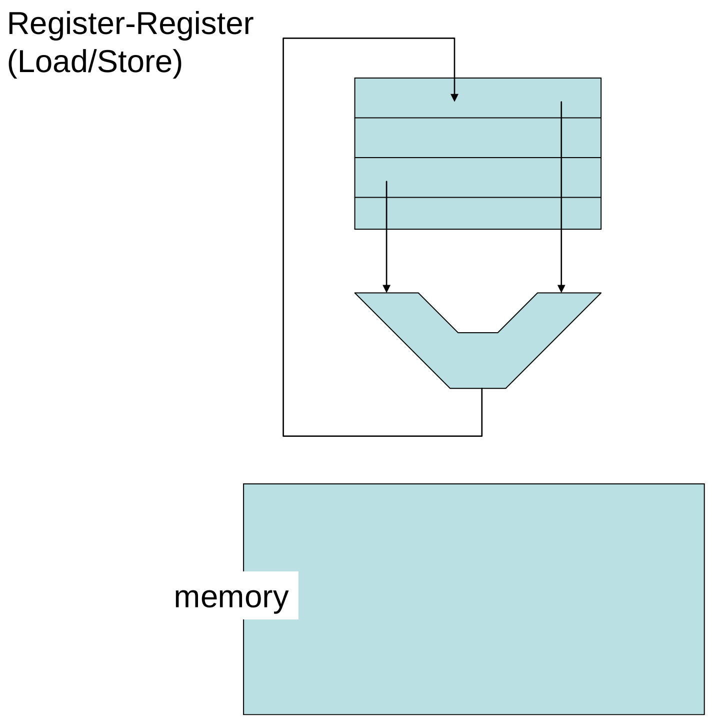
Encode Assembly statements to some type of bytecode: Make your **instructions fixed length**

Op Code Operand 1 Operand 2

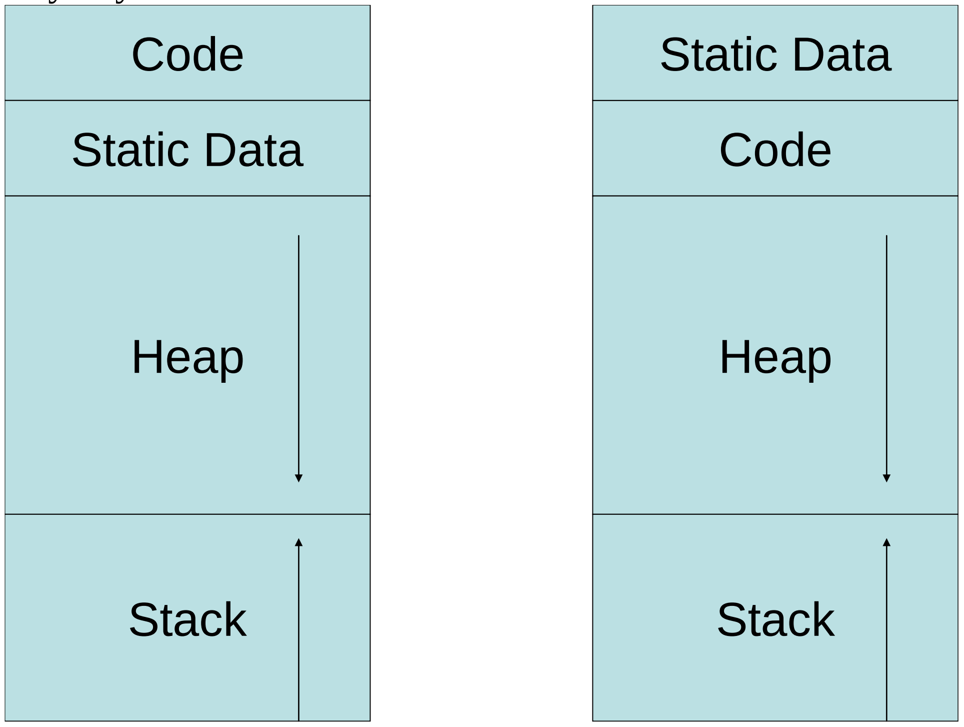
I strongly suggest you use an integer for the Op Code, Operand 1 and Operand 2 thus your fixed size instruction will be 12 bytes long. This is NOT optimal but you don't need optimized solutions just a good workable solution.

Directives (Labels) are NOT placed directly into memory. Only the data is placed into memory!!!!!!!

Execute Byte code Program (Virtual Machine)



Real Memory Layout



Pick one solution or the other Don't Mix them together that is not a solution. That is a MESS!

- Starting with Code first makes it easy to determine your initial PC (PC=0).
- Starting with Static Data first seems to be more readable than the inverse.

Just make a choice a stick to it! Don't waffle that just wastes time.

Virtual Machine

NOTE: Type Cast as allowed by C can be very helpful here!!!

MemoryType mem[MEM_SIZE];

- Static Data (Byte or 4 byte Int)
- Bytecode
- Heap
- Stack

ByteCode IR;

- Object or Structure
    - opCode is the instruction to execute.
    - opd1 is the first operand of the instruction.
    - opd2 is the second operand of the instruction.
- PC is an index into mem[].

RegisterType reg[REG_SIZE];

- Integer
- R0 is 0 & R1 is 1
- Encoding registers as an index into reg[].

The Big Switch (VM)

```
PC = Beginning_Address;
Running = True;
# this pseudo code is not intended to execute
while(Running) {
    IR = mem.fetch(PC);  # fetch the current instruction from memory
    switch(IR.opCode) {
    case ADD:
        reg[IR.opd1] = reg[IR.opd1] + reg[IR.opd2];
        PC++;
        break;
    …

    case MOV:
        reg[IR.opd1] = reg[IR.opd2];
        PC++;
        break;
    …

    case LDR:
        reg[IR.opd1] = mem.getInt(IR.opd2);
        PC++;
        break;
    …

    }
}
```