CSC 214

Two Pass Assembler

Assigned 03/17/14

Due 04/07/14

Purpose: Implement a two pass assembler with pseudo-operations, expressions and error handling to create machine code output as specified below. Work in teams of two or three as indicated in class.

Architectural Design:

12-bit words
128-word memory (12 bits/word) word addressable
Stack (last memory down) + 3 Temps (12 bits wide) – inaccessible from ASM
7-bit program counter (PC)
condition code (2 bits) : = or zero(00), > or positive(01), < negative (10)
machine instruction cycle – fetch, decode, execute, (writeback)

Machine instructions and operation codes: (* indicates affects condition code; ** indicates affected by condition code)

Inst.	Code	mnemonic (Format – all F1, see below)
push X*	00000	PSH – push value at memory X onto stack
pop X*	00001	POP – pop top of stack to memory X
add/sub*	00010/00011	ADD, SUB
and/or.*	00100/00101	AND, IOR
not*	00110	NOT
testskip**	00111	TSS – skip next instruction if zero CC
shift(l/r)*	01000/01001	SHL/SHR – shifts word on top of stack (6 bits).
jump **	01010-01101	JMP (no **), JEQ, JGT, JLT
inc X/outc X	01110/01111	INC/OTC – input/output a character (right-most 6 bits in word, convert to binary)
ini X/outi X	10000/10001	INI/OTI – input/output an integer, convert to binary
halt	11111	HLT

Machine instruction format:

Format (F1) _____ ___________
5 7

Hardware:

Memory organization – linear, 128 words : 12 bit word size 2's compliment
Stack – memory based grows from top down
Operations - 3 temporary registers (pop, pop, op, push or pop op push)
Data types/formats: Integer (12 bits), Character (6 bits - BCD)
PC – program counter (7-bits)
Flags – condition code register (00, 01, 10)
Instruction register – 12 bits (5 bit op code, 7 bit address).

Assembly language instruction format:

label – alphanumeric, 6 characters maximum, left justified in columns 1 to 6
column 7 - blank
operation mnemonic – as developed in program 1 (columns 8-10): PSH, etc.
column 11 - blank
operand – specified by at most 6 characters, left-justified, col 12-17.
column 18+ blank, or comment (18 blank, 19 to end of line comment)

Addressing mode notation

memory – symbolic (any valid label), numeric (denoted by b, d, x), unsigned (0-127)

Pseudo-ops

notation to distinguish op vs. pseudo-op (.)

mnemonics:	.nd	end of assembly listing.
	.dw	define integer word with quantity
	.ds	define storage (followed by a number of words to reserve)
	.dc	define character (right most 6 bits)
	.st	start of assembly listing

these are ASM directives, they do NOT create code nor do they occupy memory.

Errors

invalid operation code
wrong number of operands

For these cases, the assembly language instruction should be printed on the output in the usual way, but the location counter is not advanced and no code is to be generated. Flag the offending instruction by placing question marks (???????) in the field that would ordinarily contain the machine language instructions.

undefined symbols
multiply defined symbols

If an operand is undefined, do generate the code for that instruction, but print the characters "UUUUU" in the offending field. Similarly, if an operand is multiply defined, print "MMMMM" in the operand field.

Program Input: ASM program – input, add two numbers (integers), output result

```
        .st
        INI X  /Input and save to X (decimal input-convert to binary)
        INI Y  /Input and save to Y
        PSH X          /Frist addend
        PSH Y          /Second addend
        ADD            /Result – pop, pop, op, push
        POP Z          /Save result
        OTI Z  /Output result (convert to decimal)
        HLT
X,      .dw    0
Y,      .dw    0
Z,      .dw    0
        .nd
```

Code Generation:

```
100000001000
100000001001
000000001000
000000001001
000100000000
000010001010
100010001010
111110000000
000000000000
000000000000
000000000000
```

Program Output:

Print the program **listing**, including the assembler language program and the corresponding machine language code (binary), side by side (max of 50 lines of assembly language instructions). You may also output the machine code in hex (for checking).

Print the regular symbol table with columns for the symbolic name and the corresponding location counter value (max of 50 symbols).

Print a table of undefined symbols, with columns for the symbolic name and the line numbers on which that symbol was used as an operand (max of 5 references to an undefined symbol).

Print the table of multiply defined symbols with columns for the symbolic name and the line numbers on which that symbol appeared as a label (max of 5 definitions of a symbol).

If the program is correct, output the **machine code** to a text file (proj1.mc)

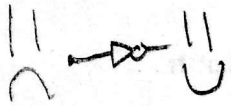
Data: Use proj1a.dat in the instr directory to test your program.

Turn in: You should turn in your program via the submit command (text only). Also you should turn in a report including a cover page (name, team, date), a summary of your work including a statement of the problem, any difficulties encountered (or parts not working), and any additional observations including the time spent by each team member. A complete listing and a sample execution with the data provided should be included as an appendix with the account name and file listings used for the submit process.

Hints: Build you program incrementally. Design and implement your data structures. Get the operations working with simple stubs and then fill in the details. For example, process the operations first, add the operands, and then any special features. Use simple error handling flags and then fill in the tables. Focus on functionality and make sure each team member understands the program.

BCD: An older 6 bit data representation format: note that each character is two groups of 3 binary digits long rather than 4 as in the byte. In the table below, space is a 020, 9 is a 011, etc. where the leading 0 represents an octal number (base 8 – zero to 7). By using 6 bits, we can keep two characters in one (12 bit word – a form of packing) and use the shift instructions to resolve individual characters or waste some space and keep each character in the right-most 6 bits of each word.

	000	001	002	003	004	005	006	007
000	0	1	2	3	4	5	6	7
001	8	9	[	#	@	:	>	?
002		A	B	C	D	E	F	G
003	H	I	&	.	]	(	<	\
004	^	J	K	L	M	N	O	P
005	Q	R	-	$	*	)	;	'
006	`	/	S	T	U	V	W	X
007	Y	Z	<	,	%	=	"	!
	000	001	002	003	004	005	006	007



Please note that there are many 6-bit BCD variants so use the table above. So, for example a 1 would be stored as 000001, a K as 100010, etc.

NOTE: ALTHOUGH I HAVE TRIED TO BE AS CAREFUL AS POSSIBLE IN THE DOCUMENTATION, IT IS EASY (AS YOU WILL DISCOVER) TO MAKE AN ERROR ON LONG STRINGS OF 0'S AND 1'S, ETC. SO IF THERE ARE ANY ERRORS, IT IS UP TO YOU TO RESOVLE ANY PROBLEMS BEFORE AN ASSIGNMENT IS DUE. ANY CONFUSION SHOULD BE CLEARED UP IN CLASS, OTHERWISE I WILL EXPECT YOU TO MAKE THE CORRECT INTERPRETATION ACCORDING TO ME. GET STARTED EARLY, DON'T WAIT TO THE LAST MINUTE – YOU WILL LOOSE POINTS!!!!!