

Machine Description

We will be developing software for the **FICKELL-Functionally Integrated Computer Key Entry Level Language**, There are 4 pages of 4096 words (32 bits each) of pc-relative addressable memory. The lab problems will center on this computer. The development platform and language for this software is a choice to be made by your team. To avoid coding conflicts I recommend that each member use the same version of the language. On the next several pages you will find the specifications. Each machine or software specification is assigned a Reference code that will be used while documenting your software.

Page 0 Active page: Instructions can only be executed from this page. Therefore if you want to execute instructions on page 3 in memory you must copy the entire page to page 0.

Page 1 to 3 Data/Instruction storage page: Instructions and data can be stored here waiting for the opportunity to be moved to the active page. An instruction can fetch or store results directly into any of the pages.

Specifications Hardware

Ref.	Item	Description
H1	Processor	Lincoln 1.0 10.2 GHz
H2	Word size	32 bits
H2	Memory	4096 words. The memory is word (not byte) addressable MEM(0:3,0:4095) .
H3	Registers	Total of 15
H3.1	Arithmetic Registers	8 for mathematics REG(0:7)
H3.2	Index Registers	7 for addressing & indexing through memory XREG(1:7)
H4	Arithmetic Unit	Two's compliment arithmetic. Note: Rely on the arithmetic operations on the development system. Do not try to do your own version of bit arithmetic.
H5	Internal Representations	32 binary digits. For numbers the left most bit is a sign bit. Character strings use all 32 bits for a total of four characters.
H5.1	Numbers	Each 32-bit word, when interpreted as an integer decimal number is in the range -2^{31} to $2^{31}-1$. It corresponds to 8 columns (right justified, with minus sign if negative, and lead blanks) of the external media.
H5.2	Characters	The ASCII (8 bit) codes are used for character information. Since each 32 bit word can contain four characters.
H6	Addressing	The machine is word addressable

Instruction Specifications

All instructions are 32 bits in length with seven different formats. The instructions have the formats:

F.1 **opc r,s(x)**

op code 6 bits	r 3b	x 3b	unused 2b	pg 2b	unused 4 bits	s 12 bits	32
-------------------	---------	---------	--------------	----------	------------------	--------------	----

F.2 **opc r,s**

op code 6 bits	r 3b	unused 11 bits	s 12 bits	32
-------------------	---------	-------------------	--------------	----

F.3 **opc s or none – if none the s field is also unused**

op code 6 bits	unused 14 bits	s 12 bits	32
-------------------	-------------------	--------------	----

F.4 op c s(x)

op code 6 bits	unused 3b	x 3b	unused 2b	pg 2b	unused 4 bits	s 12 bits	32
-------------------	--------------	---------	--------------	----------	------------------	--------------	----

F.5 op c dev,s(x),type,length

op code 6 bits	dev 3b	x 3b	type 2b	pg 2b	length 4 bits	s 12 bits	32
-------------------	-----------	---------	------------	----------	------------------	--------------	----

F.6 op c pg1,pg2

op code 6 bits	unused 6b	pg2 2b	pg1 2b	unused 4 bits	unused 12 bits	32
-------------------	--------------	-----------	-----------	------------------	-------------------	----

F.7 op c r1,r2

op code 6 bits	r1 3b	r2 3b	Unused 20 bits			32
-------------------	----------	----------	-------------------	--	--	----

Syntax Specifications

Ref.	Feature	Requirement
S1	Instructions	All instructions are alpha characters. Can be mixed case
S2	Instruction Syntax	The assembler must verify that the input it receives is valid. The input must meet all syntax rules.
S2.1	Labels	Labels in this machine are 1 to 32 columns in length. They can be upper or lower case or a combination. A label must start with an alphabetic character (A-Z, a-z). Columns 2-32 can be 0-9, A-Z, or a-z. We will limit the programmer to 50 labels per assembly. Labels are case dependent.
S2.2	Numbers	The numbers in instructions are integers in the range 0-4095
S2.3	Characters	Items that can be enter directly through the keyboard with in a single stroke or also while holding down the shift key.
S2.4	Instruction	See Instructions. Alpha characters
S2.5	Operands	Alpha or numeric. Once the operand field is started there can be NO embedded blanks.
S2.5.1	Register Specification	Register field references arithmetic registers 0 to 7 The Field can also contain a previously equated label in the same range.
S2.5.2	Memory Reference	The Assembler supports a variety of memory references
S2.5.2.1	rlabel	Symbolic memory reference within the program
S2.5.2.2	exlabel	Symbolic memory reference outside the program
S2.5.2.3	nreference	numeric memory reference within the program
S2.5.3.6	Literal/Immediate	Instructions that normally reference memory address locations may reference numeric literals instead. Literals have the following format: ADD 7,=153. The use of the LITERAL =153 will cause the assembler generate a word at the end of the program. Literals can range -2^{31} to $2^{31}-1$. Literals can also be hex or binary. 32H or 32h 1010B or 1010b
S2.5.3.7	* (i.e. Star or asterisk)	* or *+number or *-number (number ranges 0 to 4095 Use * as the current instruction address during assembly e.g. LDR 1,* would mean to place the current location counter in the last 12 bits of the instruction. LDR 1,*+ll would mean to place the result of the LDR 1,*-ll arithmetic operation in the last byte of the instruction

S3.0	Comments	Alpha numeric string
S3.1	Full Line	A line of comment begins with a : in column 1. This line is printed on the assembly listing.
S3.2	End of Line	If a comment is needed on an instruction of Directive you must use a : to separate the comment from the operand field.
S4.0	Field separators	Between label and opcode separator is a blank or tab Between operands the separator is a single comma

Directives Specifications

DS	Directive Syntax	format
DS1	Label	A variety of options are supported.
DS1.1	rlabel	string of 2 to 32 characters and numbers meeting syntax rules for a label
DS1.2	none	
DS1.3	nlabel	label, number 0 to 4095, or external reference, or elabel
DS2	Directive name	Alpha string. Mixed case
DS3	Operands	
DS3.1	number	Range 0-4095
DS3.2	rlabel	string of 2 to 32 characters and numbers meeting syntax rules for a label
DS3.3	eXlabel	string of 2 to 32 characters and numbers meeting syntax rules for a label. Representing a reference to a label in a totally different module.
DS3.4	dmax	Directive range numeric 0 - 4095
DS3.5	none	
DS3.6	Expression	See Notations and Their Meanings
DS3.7	Star	* or *+number or *-number (number ranges 0 to 4095

Directives

Ref	Directive	Format	Description	LC
D1	STT	rlabel STT p,pmax	stt Directive sets the assembler LC to the relocatable address specified. The label on the serves as the module name. Must appear first in the program input file. The start can only have a numbers in the operand field.	N
D2	RESET	rlabel reset p,nnn or elabel PG=p LC=nnn nnn can not exceed dmax	reset Directive sets the assembler LC to the relocatable address specified. There can be more than one per assembly. Generates a Linking record in the object file. Changes LC value. Does not generate lines of code.	Y ¹
		NOTE on RESET	¹ No specific word is generated. The result of this directive is to increase the LC by the base-10 number specified.	
D3	EQU	rlabel equ dmax or elabel or *	equate rlabel to the operand. If the operand is a label then that label must have been previously equated. Expressions are allowed. Star notation is allowed.	N
D4	EQUe	rlabel eque expression	See Notations and Their Meanings	N
D5	ENT	None ENT rlabel,rlabel, rlabel,rlabel Note: up to 4 operands	ENT variable (entry) name - declares the symbol to be an entry name that must appear as a label somewhere in this program. This symbol may be used as an operand by other programs. Each entry generates a Linking record in the object file.	N

D6	EXT	None EXT rlabel,rlabel, rlabel,rlabel Note: up to 4 operands	external variable name - declares the symbol to be an external name. rlabel must not appear as a label in this program. rlabel may be used as an operand by this program. The symbol must be defined in some other program by a entry	N
D7	end	none end rlabel	end of source program. This directive is used to tell the assembler that all the input has been processed. Any lines after this Directive should generate a warning message. rlabel must match pgn_name	N
D8	AEXS	rlabel AEXS rlabel or dmax	Alternative Execution Start	
D9	SKS	ol SKS dmax	SKip Storage	Y
D10	chc	ol CHC dmax*'cccc'	CHaracter Constant	Y
D11	num	ol num dmax*nmax	Base 10 number in the range -2^{31} to $2^{31}-1$	Y
D12	adc	ol adc rlabel or elabel or *+constant or *-constant	address constant . The address of the operand is placed in the last 12 bits of the memory word. Note: this directive is always relocatable or external.	Y
D13	adce	olabel adce expression	See Expressions Specifications	Y
D14	NOP	none NOP none	No operation inserts a NOP \leftrightarrow SWR 0,0	Y

Instructions: Arithmetic

IA1	000000	ADD	R,Memory(X)	ADD reg & memory	$c(R)=c(R)+c(S(X))$
IA2	000001	SUB	R,Memory(X)	SUB tract reg & memory	$c(R)=c(R)-c(S(X))$
IA3	000010	MPY	R,Memory(X)	MultiPLY reg & memory	$c(R)=c(R)*c(S(X))$
IA4	000011	DIV	R,Memory(X)	DIV ide reg & memory	$c(R)=c(R)/c(S(X))$
IA5	000100	SQV	R,S	SQ uare Value	$c(R)=c(R)**S$
IA6	000101	ADDR	R,R	ADD Reg & Reg	$c(R)=c(R)+c(R)$
IA7	000110	SUBR	R,R	SUB tract Reg&Reg	$c(R)=c(R)-c(R)$
IA8	000111	MPYR	R,R	MultiPLY Reg&Reg	$c(R)=c(R)*c(R)$
IA9	001000	DIVR	R,R	DIV ide Reg & Reg	$c(R)=c(R)/c(R)$

Instructions: Storage and Retrieval

IS1	011000	LDR	R,Memory(X)	LoaD ReGister	$c(R)=c(S(X))$
IS2	011001	LDN	R,Memory(X)	LoaD Register Negative	$c(R)=-c(S(X))$
IS3	011010	STR	R,Memory(X)	SToRe Register	$c(S(X))=c(R)$
IS4	011011	CLR	R,Memory(X)	CLear Register	$c(R)=0$; $S(X)$ is ignored
IS5	011100	CLRA	none	CLear all Arithmetic registers	R0-7 equal zero
IS6	011101	CLR X	none	CLear all iXdex registers	X1-7 equal zero
IS7	011110	SWR	R1,R2	SW itch Registers	$c(R1)=c(R2)$, $c(R2)=c(R1)$

Instructions: Shift/manipulate

IM1	010000	SRL	R,Memory(X)	Shift Right Logical	Shift right S(X) Places fill with zeros
IM2	010001	SLL	R,Memory(X)	Shift Left Logical	Shift left S(X) places fill with zeros
IM3	010010	SRA	R,Memory(X)	Shift Right Arithmetic	Shift right S(X) places, fill with sign bit
IM4	010011	SLA	R,Memory(X)	Shift Left Arithmetic	Shift Left S(X) Places fill with sign bit
IM5	010100	ROL	R,S	ROtate Left	Rotate S bits left Including sign bit
IM6	010101	ROR	R,S	ROtate Right	Rotate S bits right Including sign bit

Instructions: Logical

L1	001001	ANL	R,S(X)	And Logical register	$c(R)=c(R) \text{ AND } c(S(X))$ logical
L2	001010	ORL	R,S(X)	OR Logical register	$c(R)=c(R) \text{ OR } c(S(X))$ logical
L3	001011	ANA	R,S(X)	And Arithmetic register	$c(R)=c(R) \text{ AND } c(S(X))$ Arithmetic
L4	001100	ORA	R,S(X)	OR Arithmetic register	$c(R)=c(R) \text{ OR } c(S(X))$ Arithmetic

Instructions: Jump, Test and Control

JT1	100000	JEQ	R,S(X)	Jump EQual Zero	if $c(R)=0$ then $LC=S(X)$
JT2	100001	JLT	R,S(X)	Jump Less Than Zero	if $c(R) < 0$ then $LC=S(X)$
JT3	100010	JGT	R,S(X)	Jump Greater Than Zero	if $c(R) > 0$ then $LC=S(X)$
JT4	100011	JMP	R,S(X)	JuMP Unconditionally	$LC=S(X)$, R is ignored
JT5	100100	JLK	R,S(X)	Jump and LiNK	$c(R)=LC+1$, $LC=S(X)$
JT6	100101	JDR	R,S(X)	Jump or Decrement Register	$c(R)=c(R)-1$, if $c(R)=0$ then $LC=S(X)$
JT7	100110	JEQX	X,S(X)	Jump Index reg EQual Zero	if $c(X)=0$ then $LC=S(X)$
JT8	100111	JNEQX	X,S(X)	Jump Index reg NOT EQual Zero	if $c(X) \neq 0$ then $LC=S(X)$

Instructions: INDEX REGISTERS

X1	111000	LDX	X,S(X)	LoAD Index register	$c(X)=c(S(X))$
X2	111001	ADDX	X,S(X)	ADD Xreg & memory	$c(X)=c(X)+c(S(X))$
X3	11010	CLRX	X,S(X)	CLear iXdex	$c(X)=0$ DELETED

Instructions: Control

C1	111101	SWP	PG1, PG2	SW itch to Alternate Page	Move contents of memory page PG1 into page PG2, PG2 into PG1
C2	111110	DMP	1, 2 or 3	DuMP Memory, LC, MEM(LC), and registers	S=1 then Level 1 Dump S=2 then Level 2 Dump S=3 then Level 3 Dump
C3	111111	HLT	dmax	HaLT program execution Display code to screen	Halt execution

Instructions: Input/Output

IO1	101000	WDV	D, S(X), T, L	W rite to DeV ice L bytes from S(X) as type T D=1 (screen); D=2 (disk file) T= 1 (signed integer); 2 (character)
IO2	101001	RDV	D, S(X), T, L	R ead from DeV ice L bytes into S(X) as type T D=1 (keyboard); 2 (disk file) T= 1 (signed integer); 2 (character)

Notations and Their Meanings

Notation	Meaning and Ranges
elabel	previously equated label
nlabel	label, number 0 to 4095, or external reference, or elabel
rlabel	required label
olabel	optional label
exlabel	External Label reference
cccc	1 to 4 characters in single quotes
n	0-9
dmax	0-4095
nmax	-2^{31} to $2^{31}-1$
p	Page number 0,1,2 or 3
pmax	Word position on the page 0 to 4095
none	no label is expected in this field. If a label is present it must be considered a warning level error
C(?)	Indicates that the contents of a register or memory location is to be used C(R) contents of the register C(S(X)) contents of the memory location
S(X)	Indicates that you should use the value of the bit string in the S portion of the instruction + the contents of the index register + pg*4095

Expression Specifications

Ref.		
EX1	Operand_Expression	begins with a * and is followed by a single + or - with the next operand being a constant or previously equated symbol. The result of the computation must be in the range 0 to 4095. (can not use in conjunction with external references) e.g. CNTE GOTO , *+5 -ADD 1,*-1-
EX2	EQU_Expression	The operands can be constants or previously equated symbols. The operators are + and -The result of the computation must be in the range 0 to 4095. (can not use in conjunction with external references) X1 EQU.e 5+2-1 or X2 EQU.e AB+CD+EF (can use local references, previously equated labels, star or constants) If there is a star in the expression it must appear first and there can only be one star per expression. Up to three operators
EX3	Adc_Expression	Appears in the AD C e. The purpose of this instruction is to perform address computations. The format includes constants, local and external references and star notation. The allowed operations are + and -. The results must fit in a 16-bit word and can be positive or negative. If there is a star in the expression it must appear first and there can only be one star per expression. adce mud+oak-dirt

Assembler Assignment (FICKELL.ASM)

You are to write a program that will interpret assembler mnemonic code by parsing, syntax checking, and translating it into valid machine code. The program must be robust enough to report errors, and generate an informative report for the end user. The program will be a 2-pass assembler as described in class. The assignment requires the development of documentation describing the machine, the assembler, the test plan, and the actual test runs. The documentation should be designed so that sections can be added to cover the loader and the simulator assignments.

Assembler Input is expected to be in character and integer (base 10 unless clearly specified as a different base) form with addresses in either absolute or symbolic form. Source code can be either fixed or floating (you must allow for either). **Label:** Must start with A-Z or a-z length cannot exceed 32 characters. Second to sixth position can be A-Z, a-z, or 0-9. **Op-code:** Can be an instruction or Directive.

Column 1 - 32	Label (optional)
33	Blank
35 - 39	Operation/Directive
40	blank
41 - n	operands starting at column 14
n+1	Comments field starting with a colon (:)

The assembler must also support, a floating format where each major component (label, opc, operands) of the instruction is separated by at least one blank. If no label is present the first column must be blank. Instructions and Directive can be in upper or lower case or a combination. All instructions and Directives are reserve words.

Assembler Output consists of three REQUIRED items. Namely, Source listing, a sorted symbol table, and an object file.

Assembler Listing or User Report

Ref:	Item	
AO1	Location assigned for this instruction and directive	Hex a decimal
AO2	Object code	4 digit hex representation of the instruction or directive
AO3	Relocation flags	A/R/E/C
AO4	Statement number	Base ten counter relative to each line of the full source program
AO5	Source Line	Actual source line provided by the programmer, with their comments

The source-listing example

```

LOC      OBJ CODE  A/R/M  STMT  SOURCE STATEMENT
(HEX)    (HEX)    flags (DEC)

pg#,0018  80029812  R     25     PT  ADD 1,AA

```

2. Machine language object file Specifications.

There are five record types: Header (type=H) contains information regarding module name, length and assumed starting address; Linking Record (type=L) come from start and entry Directives; Text (type=T) contains the hex equivalent of each word and the relocation flag; Modification (type=M) and End (type=E) record indicates the end of the module.

Object File: Header Record Specification

Ref	Item	format
OB1	Header Record	
OB1.1	H	Single character
OB1.2	:	Single character
OB1.3	Program name	string of 2 to 32 characters and numbers meeting syntax rules for a label
OB1.2	:	Single character
OB1.3	Assembler assigned program load address	6 digit hex number (page and location)
OB1.2	:	Single character
OB1.3	Total Module length	4 digit hex number (0 to 3FFF)
OB1.4	:	Single character
OB1.5	Execution Start address	4 digit hex number (0 to 03FF) must be on page 0
OB1.6	:	Single character
OB1.7	Date and time of Assembly	Julian date year,da, hh:mm:ss Jan 1 2011 10PM would be 2011001,22:00:00
OB1.8	:	Single character
OB1.9	Version number of the assembler	4 digit integer
OB1.18	:	Single character
OB1.19	FICKELL-ASM	11 character string
OB1.20	:	Single character
OB1.21	Program name	Same as OB1.3

Object File: Linking Record Specification

Ref	Item	format
OB2	Linking Record	
OB2.1	L	Single character
OB2.2	:	Single character
OB2.3	Entry Name	string of 2 to 32 characters and numbers meeting syntax rules for a label
OB2.4	:	Single character
OB2.5	Location within this pgm	6 digit hex number (page and location) pghhhh all 6 digits are required all upper case
OB2.6	:	Single character
OB2.7	Program Name	string of 2 to 32 characters and numbers

Text Record Specification

OB3	Text Record	
OB3.1	T	Single character
OB3.2	:	Single character
OB3.3	Program assigned location	6 digit hex number (page and location) pghhhh all 6 digits are required all upper case
OB3.4	:	Single character
OB3.5	Instruction/data word	8 digit hex code all 8 digits are required all upper case
OB3.6	:	Single character
OB3.5	Address status flag	A, R or M
OB3.6	:	Single character
OB3.7	Number of M adjustments required	1 digit hex number
OB3.8	:	Single character
OB3.9	Program Name	String of 2 to 32 characters and numbers meeting syntax rules for a label

Object File: Modification Object Record

Ref	Item	format
OB4	Modification Record	
OB4.1	M	Single character
OB4.2	:	
OB4.3	Program location for address modification	6 digit hex number (page and location) pghhhh all 6 digits are required all upper case
OB4.4	:	Single character
OB4.5	Adjustments	Up to 4 sets
OB4.7.1	:	Single character
OB4.7.2	Sign	Plus or minus + or -
OB4.7.3	:	Single character
OB4.7.4	Label name to be used as index to linker symbol table	string of 2 to 32 characters and numbers meeting syntax rules for a label
OB4.8	:	Single character
OB4.9	Program Name	string of 2 to 32 characters and numbers meeting syntax rules for a label

End Record Specifications

OB5	End Record	
OB5.1	E	Single character
OB5.2	:	Single character
OB5.3	Total number of records in the object file	4 digit hex number
OB5.4	:	Single character
OB5.5	Total number of linking records	4 digit hex number
OB5.6	:	Single character
OB5.7	Total number of Text records	4 digit hex number
OB5.8	:	Single character
OB5.9	Total Number of Modification records	4 digit hex number
OB5.10	:	Single character
OB5.11	Program Name	string of 2 to 32 characters and numbers meeting syntax rules for a label

Loader File: Header Record

Ref	Item	format
LM1	Header Record	
LM1.1	H	Single character
LM1.2:	:	Single character
LM1.3	Module name from first object	string of 2 to 32 characters and numbers meeting syntax rules for a label
LM1.4	:	Single character
LM1.5	Combined module load address	6 digit hex number (page and location) pghhhh all 6 digits are required all upper case
LM1.6	:	Single character
LM1.7	Module Execution start address	4 digit hex number must be on page zero
LM1.8	:	Single character

LM1.9	Total length of combined module	4 digit hex number
LM1.10	:	Single character
LM1.11	Date and time of Assembly	Julian date yearday, hh:mm:ss Jan 1 2011 10PM would be 2011001,22,00,00
LM1.12	:	Single character
LM1.13	FICKELL-LLM	11 character string
LM1.14	:	Single character
LM1.15	Version number of the assembler	4 digit integer
LM1.16	:	Single character
LM1.17	Program Name	string of 2 to 32 characters and numbers meeting syntax rules for a label

Load Module: Text File

LM2	Text Record	
LM2.1	T	Single character
LM2.2	:	Single character
LM2.3	Program assigned location	6 digit hex number (page and location) pghhhh all 6 digits are required all upper case
LM2.4	:	Single character
LM2.5	Instruction/data word	8 digit hex number
LM2.6	:	Single character
LM2.7	Program Name	string of 2 to 32 characters and numbers meeting syntax rules for a label

End Record Specifications

OB5	End Record	
OB5.1	E	Single character
LM5.2	:	Single character
LM5.3	Total number of Records	4 digit hex number
LM5.4	:	Single character
LM5.5	Total number of Text Records	4 digit hex number
OB5.6	:	Single character
OB5.7	Program Name	string of 2 to 32 characters and numbers meeting syntax rules for a label

Sample Errors: If you find an error you will need to print an error message below the source line with the error and make a decision as to what action to take. **Some errors you will discover include:**

Errors	Warning, Serious, Fatal	Action
EF.1	Invalid start directive	Fatal error: Assembly has been stopped
EF.#	First line not a valid STT	Syntax requires the first line to be a valid STT
EF.#	LC is now larger the 4095	Your program must use less than 4096 words of memory on a page
EF.#	Exceeded the maximum number of symbols	use less labels!
EF.#	Exceeded the maximum number of lines of code	make your program more concise
ES.1	Invalid function code	User has entered an invalid function code.
ES.#	Invalid Label, label ignored processing continues	Labels must start with A-Z, a-z
ES.#	Operation code invalid, NOP substituted	Use one of the valid instructions

ES.#	S-field invalid	must be a valid defined label NOP substituted
ES.#	Invalid Literal	NOP substituted Literal must be in range -2^{31} to $2^{31}-1$
ES.#	Too many comma's in the operand field	NOP substituted remove erroneous comma's
ES.#	Directive not in proper range, line ignored	operand must be in the range 0 to 4095 or an equated symbol in the range 0 to -2^{31} to $2^{31}-1$. If adc must be a label define in this pgm
ES.#	Directive requires a label	line ignored
ES.#	Invalid data value in num Directive,	num 0 substituted if numeric must be in range -2^{31} to $2^{31}-1$
EW.1	Blank Line	Ignored, no message
EW.#	operand field for ent/ext is invalid	line ignored
EW.#	Label is invalid	field ignored
EW.#	Invalid comment field	line ignored
EW.#	Blank line found	line ignored

Note: This by no means is an all-encompassing list of the errors you will have to be prepared to capture and report on.

Usability Requirements

The user must be able to use a command line to indentify the program to be run, location of the input file or files, destination for the output file, and flags to indicate any user selectable options (such as DEBUG or Display Pass 1 intermediate version). In addition to the command line you may also develop a web based GUI to run the program, this is your choice.

Keep in mind that the graders of you labs will install your software on their machines or CSE lab machines.

Java Based	Windows, MacIntosh or Unix
C, C++	Windows
Resolve	Windows

The Labs for CSE560N

SP0: You are to meet with the graders and share your design and a few sample runs prove that your team can parse a valid instruction line, compute the location counter for each line, and build a symbol table. These will be valid instructions and two directives start and end.

Requirements: WEB access to

Draft of the documentation

Testing Philosophy

Output:

Display the original source line

A break down of the components along with LC.

A formatted Symbol Table (unsorted)

SP1: Full Pass 1. You will present in a real time grading session your pass 1, complete versions of your manuals and to participate in a code review with the graders. Your software will be initially graded through a real-time grading session.

Requirements: WEB access to

Final documentation

Detailed Test plan

Turn in all test plan runs using the same source code

OUTPUT: Assembly Report – formatted Intermediate Version

Display the original source line

Display errors below the instruction line containing the error

Display a formatted Symbol Table (sorted)

SP2: Full Assembler. The software must be a fully integrated pass 1 and 2. The code must handle all aspects of the language report errors, and create all the reports. Your software will be initially graded through a real-time grading session. Sample test program is below.

Requirements: WEB access to

Final documentation with additional information and adjustments due to SP2

FINAL Detailed Test plan

Turn in all test plan runs using the same source code

OUTPUT: Assembly Report

Display the original source line

Display errors below the instruction line containing the error

Display a formatted Symbol Table (sorted)

An Object File,

SP3: LINKER Assignment. You are to write a linker (FICKELL.LLM) which will read in object files produced by the assembler, link these files together, handle relocation, generate a load file (Simulator input format). FICKELL.LLM is also to print a load map showing the names and memory location of all the modules and the ENT points for each module. The output must include: Load Module, Load Map, listing of the object deck as it entered the loader program, and a list of the errors being reported.

Requirements: WEB access to

Final documentation with additional information and adjustments due to SP2

FINAL Detailed Test plan

Turn in all test plan runs using the same source code

OUTPUT: Display the hex code as it is read in

Display error messages immediately below the hex that caused the error

Display a Load Map--Loader symbol table

Display a Load File--suited for lab 4 input

SP4: Simulator Assignment FICKELL.SIM. You are to write a program that will simulate the operation of the FICKELL hardware. The simulation should be done at a high level. You do not have to simulate 2's complement binary arithmetic at the bit level. To do an multiply you simply:

ADD 1, C4	Pseudo-code temp=REG(1)*mem(C4) Test temp for overflow Reg(1)=temp
-----------	---

Requirements: WEB access to

Final documentation with additional information and adjustments due to SP2

FINAL Detailed Test plan

Turn in all test plan runs using the same source code

OUTPUT: Display the hex code as it is read in
Display error messages immediately below the hex that caused the error
Display a Dump of active memory
Results of the actually program

DUMP Format:

Level 1 All 15 registers, current LC value and Current Instruction

Level 2 Memory from start of program to end of program,

Level 3 Full print (items 1 and 2)

DUMP Format:

LC pg,lc WORD=HHHHHHHH

R0=HHHHHHHH R1=HHHHHHHH R2=HHHHHHHH R3+HHHHHHHH R4+HHHHHHHH

R4=HHHHHHHH R5=HHHHHHHH R6=HHHHHHHH R7+HHHHHHHH

XR1=HHHHHHHH XR2=HHHHHHHH XR3+HHHHHHHH XR4+HHHHHHHH

XR4=HHHHHHHH XR5=HHHHHHHH XR6=HHHHHHHH XR7+HHHHHHHH

Dump Format: CODE 2

```
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
pg,lc HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH HHHHHHHH ..... HHHHHHHH HHHHHHHH
```

where:

pg,lc represents memory offset in hex HH,HHH

H represents a hex number

DEBUG Format:

```
---Before Simulation--- ---Before Simulation--- ---Before Simulation---
PG=HH LC=HHHH MEM(LC)= bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb opc: bbbbb MEM(LC)=HHHHHHHH
values: S=nnnn MEM(S)=nnnnnnnnnn reg(R)= nnnnnnnnnn Xreg(X)= nnnnnnnnnn
---Before Simulation--- ---Before Simulation--- ---Before Simulation---
```

any output generated: error message or results of the output commands

```
===After Simulation=== ===After Simulation=== ===After Simulation===
values: S=nnnn MEM(S)=nnnnnnnnnn reg(R)= nnnnnnnnnn Xreg(X)= nnnnnnnnnn
===After Simulation=== ===After Simulation=== ===After Simulation===
```

where:

b represents a binary string

H Hex

Note: you need to print the "variable=" as per the examples.

Appendix ALT1: Assembler Test Program.

The program is supposed to read four pairs of variables XX and YY, and compute the sum of their corresponding quotients, XX/YY. Program has not been validated. Clean up any syntax errors and verify the logic.

```

ALT1  STT    1,4                      :Revised Program 10/12/11
      EXT    ZX
P1    LDR    0,=0                      :CLEAR SUM TO ZERO
      LDR    2,=0                      :CLEAR SUM TO ZERO
      LDRX   1,=4                      :SET INDEX TO ZERO
P2    RDV    0,XX(1),1,6               :READ NEXT XX
      WDV    1,XX(1),1,6               :ECHO XX
      RDV    0,YY(1),1,6               :READ IN NEXT YY
      WDV    1,YY(1),1,6               :ECHO YY
      LDR    2,XX(1)                   :FORM XX/YY
      LDR    3,YY(1)
      JEQ    3,ER                      :IF DIVIDE BY 0, GO PRT MSG AND HALT
      DIV    2,YY(1)                   :OTHERWISE DIVIDE
      STR    2,QQ(1)                   :STORE RESULT
      ADDX   1,=-1                      :DECREMENT INDEX REGISTER
      JEQX   1,DONE                    :Increment Count
      ADD    0,QQ(1)
      JLT    2,P2                      :IF NOT DONE, LOOP BACK
DONE  STR    0,Q1
      WDV    1,M1,2,6
      WDV    1,Q1,2,4
      JMP    3,E1                      :THEN GO TO THE EXIT ROUTINE
ER    WDV    1,ME,2,12                 :ZERO DIVISOR, PRINT 'Y IS 0 '
      DMP    0,0                       :DUMP
      HLT    0,0                       :HALT
E1    WDV    1,M2,2,12
      LDR    1,A1
      JLK    2,0(1)
:  CONSTANTS AND TEMPORARIES
A1    ADC    ZX                        :ADDRESS OF External Reference
C1    NUM    1
C4    NUM    4
ME    CHC    'Y IS'                    :MESSAGE IS 'Y IS 0 '
      CHC    ' 0'
      CHC
M1    CHC                                MESSAGE IS '      Q='
      CHC
      CHC    ' Q='
M2    CHC    'END '
      CHC    'PROG'
      CHC
QQ    SKS    4
Q1    SKS    1
TP    NUM    0
XX    SKS    4
YY    SKS    4
      END    ALT1

```

Appendix ALT2: Sample Program

Hex Loc	Instruction in Binary	Dec Loc	Symbolic Representation
lc opcode hex hex ??	binary representation of the instruction	lc dec	
04	00000011	4	NUM 17
05	00000009	6	NUM 9
--	-----	--	STT 20
14	60000005	20	LDR 0,5
16	04000009	22	MPY 0,9
18	6800000D	24	STR 0,13
1A	64800005	26	LDN 1,5
1C	0C800009	28	DIV 1,9
1E	68000011	30	STR 1,17
20	31000005	32	LDR 2,5
22	21000009	34	ADD 2,9
24	69000064	36	STR 2,108
26	61800005	38	LDR 3,5
28	25800009	40	ORR 3,9
2A	69800068	42	STR 3,104
2C	60800005	44	LDR 1,5
2E	04800009	46	SUB 1,9
30	68800068	48	STR 1,108
32	60000005	50	LDR 0,5
34	44200000	52	SLL 0,0(2)
36	68000070	54	STR 0,112
38	84000022	56	JLT 2,34
3A	E0000000	58	HLT 0,0
			END ALT2

Sample Program

Label			:Comment	LC	Hex Code	Reloc flag
ALT3	STT	0				
	LDR	1,AB	:place AB in register 1	0	00800005	r
	ADD	1,CD	:add (AB+CD)	1	80000006	r
	STR	1,RES	:store AB+CD into RES	2	68800007	r
	WDV	1,RES,1,1	:output RES	3	A0841007	r
	HLT	0	:halt program display code 0	4	FC000000	a
AB	num	10	:define data element	5	0000000A	a
CD	num	111	:define data element	6	0000006F	a
RES	num	0	:define data element	7	00000000	a
	END	ALT3	:end of program		E	