Tuesday
23/08/22

Date ___/___/___
Page ___

Compiler Design

Unit-1

1. Software → Application → providing Application to system
   System ⇒
   - compiler
   - Assembler
   - Linker
   - Loader.

Question - Difference between application and system software.

- Topic-2 Translator → convert one lang to another. (It is a system soft.)

3 types

(Hostes) ① compiler - HLL to object language (may be machine level or not)

HLL → compiler → object lang. or LL

② Interpreter (line by line) (more efficient)

HLL → [redacted] → object lang
Interpreter

③ Assembler

Assembly lang → [redacted] → OC
Assembler

ex: 01 ⇒ near to the system

MOV A,B
ADD A,B
⇒ A=B
A=A+B
} uses nearer.

near to

0 1 ⇒ system

MOV A,B
ADD A,B
Translator ⇒ Assembly lang.

Trans lator

A=B
A=A+B

Q. What is Translator
Q. What is the need of translator
Q. Explain types of translator
Q. Differentiate between compiler and Interpreter.

#ifdef, #endif

First it deals with macros (#define in C)

execution process diagram

HLL → macro procesor → Assembly compiler → *.obj → linker

*.c

linker: link all the header file, etc references.

*.exe → Loader → *.exe

.exe stored in secondary memory → primary memory.

previously it execute seperately

Wednesday 22/03/22

Topic -

Assembler

1. Specify the problem
2. Identify the data structure
3. Define the format of data structure
4. Specify algorithm
5. Look for modularity
6. Repeat step 1 to 5 for each module

① Requirement & analysis
② planning -
③ Design and analysis
④ Implementation
⑤ Testing

1. Specify the problem

Assembly lang → Assembler → object code.

mnemonic → ADD A,B
→ MOV R1, 2
} Addressing mode

- Immediate addressing mode
  Data is provided within instruction
- Direct addressing
  MOV R1, #3162
  providing the address.
- Indirect addressing mode
  MOV R1, R2
  providing address of address
  R2 = R3
- Base address
- Index address

ADD A,B

→ Generate Instruction
  → Evaluate opcode — ADD — variable size
  → Evaluate operand — A,B — fixed size
→ Evaluate pseudoops

get act as instruction to the assembler.

- Single pass algorithm
- Two pass algorithm

Output of first pass is given as input to second pass (like pipelining)

→ speed will increase — comprises multiple phases → one pass
→ complexity increase

→ Pass is one set of iteration of module
→ ~~execution of code is equal to base~~ we will combine phases into passes

Q Difference between pass and phase

	pseudoOp		Like Goto
JOHN	START	0	Labels JOHN
	USING	*,15	FOUR
	L	1,FIVE	FIVE
	A	1,FOUR	TEMP
	ST	1,TEMP	
FOUR	DC	F'4'	FOUR=4 — Define constant
FIVE	DC	F'5'	FIVE=5 — Define storage
TEMP	DS	1F	one fullword
	END		

Load Add store

Load (L) R1 = FIVE

Add(A) R1 = R1+FOUR
Store (ST) TEMP = R1

→ Register 15 act as base register
→ Load the current value of location counter in the base register.
→ It store at location 0 (START 0)

Q what is pseudoops
Q Difference between single pass and Two pass
Q what are pseudoops
Difference between DC and DS — to define the storage area
Q what is the size of fullword
now we can define a constant

F = fullword = 2 or 4 bytes. B = 1B — DW Double word = 8B
HW = Halfword = 2byte nibble = 1/2B

Thursday 01/09/22

RX — Base register — Register

DC — F'4' — number — fullword — size = 4B

size = 4 byte

we can define storage using

DS F 'a','b','c','d'

in forward — constants (just like an array)

LC → 0

R15 → 0

Conversion

			Relative Location		LC = 0, 4
JOHN	START 0				
	USING *,15				
	L 4, FIVE		0	L	1, ___ (0,15)
	A 1, FOUR		4	A	1, ___ (0,15)
	ST 1, TEMP		8	ST	1, ___ (0,15)
FOUR	DC F'4'		12	4	
FIVE	DC F'5'		16	5	
TEMP	DC 1F		20	-	
	END				1-pass

offset

symbol table

is not

* Pseudoops converted into machine code.

offset (index, Base)

pass-2

0	L	1, 16 (0,15)
4	A	1, 12 (0,15)
8	ST	1, 20 (0,15)
12	4	
16	5	
20	-	

- Symbols are stored in symbol table
- Literals are either constant or value of variables ex F'4', F'5'
- If any pseudoop is left in pass-1 it will process in pass-2

* Information that has to be maintained / stored in

1. Machine instruction. ⇒ we create machine Op Table (MOT)
2. Pseudo ops ⇒ Pseudo op table (POT)
3. Location counter ⇒ store next available location
4. Symbol table - To store information about symbols.
5. Literal table - stores literals
6. Base table - partitioned registers will act as base register

Friday 02/09/22

Date / / Page

Format of Datastructure in Assemblies

① Machine op table (MOT) → 6 bytes

Mnemonic op (Character)	Binary code (1B) Hexa-decimal	Instruction size 2 bits	Instruction format 3bits	unused 3bits
'A bb'	5A	01 → 2B	000 - RR → 2B	
'ST bb'		10 → 4B	001 - RX → 4B	
'STR b'		11 → 6B	010 → RS, 011 → SI, 100 → SS → 6B	

Pseudo op Table (3B)

Pseudo op (3B) Character	3 Bytes Address of routine to process pseudop
'START'	
'USING'	
'DROP'	

Symbol op Table (14B)

Relocatable
↑
R → Address can be changed

Symbol 8B character	Value (4B) Hexadecimal	Length 1B	Allocation 1B (Relative / Absolute) R/A → Address cannot be changed
'JOHNbbbb'	Relative loc^n (address)		
FOUR			

(address cannot be changed)

Literal are absolute

Literal op Table

Literal 8B character	Value 4B Hexadecimal	Length	Allocation (Relative / Absolute)
='4'	4 value		

Base op Table

Available 1B (Y/N) character	Content 3 Bytes Hexadecimal
0 N	
N	
15 N	
15 Y	

```
0  JOHN START 0
.
.
.
   END
```

Base table

Available (1B) (Y/N) character	Content 3 Bytes Hexadecimal
0 N	
N	
N	
14 N	
15 Y	000

8420

2 | 12
2 | 6 0
2 | 3 0
1 | 1

1100

Symbol Table

Symbol	Value	Length	Allocation
JOHN bbbb	0000	1	R
'FOUR bbbb'	0000C	4	R
'FIVE bbbb'	00010	A	R
'TEMP bbbb'	0014	4	R

Literal Table

Literal	Value
F'4'	
F'15'	

= Symbol literal

SR - Subtract

LA - Load address

Thursday 08 September '22

LC				
0	1.	PROGAM2	START 0	
0	2.		USING *, 15	
0	3.		LA 15, SETUP	
4	4.		SR TOTAL, TOTAL	2B because of R keyword
	5.	AC	EQU 2	
	6.	INDEX	EQU 3	
	7	TOTAL	EQU 4	
	8	DATABASE	EQU 13	
	9	SETUP	EQU *	
	10		USING SETUP, 15	
6	11		L DATABASE, =A(DATA1)	
	12		USING DATAAREA, DATABASE	
10	13		SR INDEX, INDEX	

BNE - Branch not equal

LC				
12	19	LOOP	L AC, = DATA1(INDEX)	
16	1A		AR TOTAL, AC	
18	1E		A AC, = F'5'	
22	1A		ST AC, SAVE(INDEX)	
26	1E		A INDEX, = F'4'	
30	19		C INDEX, = F'8000'	
34	20		BNE LOOP	
36	01		LR 1, TOTAL	
38	02		BR 14	
40	83		LTORG - Pseudops - Relocation of 8 byte will be there (Double word)	
64	24	SAVE	DS F	2000F
68		DATAAREA	EQU *	
68		DATA1	DC F'25', 26', - - -	
72			END	(2000 No.)

BR - Branch register

LTORG - Literal table organizer

Symbol Table

Symbol	Value		
PROGAM2	0000	1	A
AC	2	1	A
INDEX	3	1	A
TOTAL	4	1	A
DATABASE	13	1	A
SETUP	8	1	R
LOOP	12	1	R
SAVE	64	1	R
DATAAREA	8064	1	A
DATA1	8064	1	R

Literal table

A(DATA1)	48	4	A
F'5'	52	4	A
F'4'	56	4	A
F'8000'	60	4	A

Base table

	Base register		
store 2	BR=15	0000	
10	BR=15	6	Setup
12	BR=13	8064	DATAAREA

Questions

1. Construct tables - 18 marks
2. Identify different literals
3. various functionality of pass-1 and 2
   Draw algo of pass-1 and pass-2
   Discuss the 2-pass assembler
   What are the modules/datastructure for sucessful implementation of 2-pass assembler.

Difference between single and multipass assembler.

Friday 09/09/22

LC				
	TEST	START		
	BEGIN	BALR	15,0	2 Bytes
		USING	BEGIN+0, 15	
2		BR	4,4	
4		L	3,FF'10'	
8	LOOP	L	2, DATA(4)	
12		A	4, =F'4'	
16		BT	2, DATA(4)	
20		A	4, =F'4'	
24		BCT	3, *-16	
28		BCR	14	
30		LTORG		
	DATA	DC	F'1', 3, 3	

Branch and link register instruction to the computer to load a register with the next address branch to the address in next field

R14 ← R0

4 Bytes

2 Bytes

Symbol	value	length	Allocation
TEST	0000	1	A
BEGIN	0000	1	A
LOOP	8	1	R

Literal Table

Literal	value	length	Allocation
F'10'	38	4	
F'4'	42	4	

Base table

Base register	
BR = 15	2

Tuesday 13/09/22

Macro processor

Assembler

- macro definition
- macro call

How to define macro —

```
I/P parameter →  ARG3   MACRO
                        MACRONAME   &ARG1, &ARG2
                                    set of instruction
                        MEND
```

number of argument

```
START
    A   1, DATA1
    A   2, DATA1
    A   3, DATA1
    A   1, DATA2
    A   2, DATA2
    A   3, DATA2
END
```

```
MACRO
MACRO1  &ARG
    A   1, &ARG
    A   2, &ARG
    A   3, &ARG
MEND
```

```
START
MACRO1 DATA1
    //macro call
MACRO1 DATA2
    //macro call
END
```

```
START
LOOP1
    A   1, DATA1
    A   2, DATA2
    A   3, DATA3
LOOP2
    A   1, DATA2
    A   2, DATA1
    A   3, DATA3
END
```

```
MACRO
MACRO1  &ARG1, &ARG2,
        &ARG3
    A   1, &ARG1
    A   2, &ARG2
    A   3, &ARG3
MEND
```

```
START
    MACRO1 DATA1, DATA2,
        DATA3, LOOP1
    MACRO1 DATA1, DATA2,
        DATA3, LOOP2
        //macro call
END
```

Mapping: LOOP1, DATA1, DATA2, DATA3
MACRO1

Macro call

① positional argument

are to one mapping

&ARG4	MACRO1	&ARG1
↕	↕	↕
LOOP	MACRO1	DATA1

⑤ Keyword argument

call ⇒ MACRO1 &ARG1 = DATA1
&ARG4 = LOOP

ex:
def sum(a,b)
a+b
sum(a=9,b=5)

Conditional macros

AGP - goto

→ AIF (condition) Label
conditional branch

→ AGO Label
unconditional branch

Example

```
START
  !
LOOP1  A  1, DATA1
       A  2, DATA1
       A  3, DATA1
  !
LOOP2  A  1, DATA1
       A  2, DATA1
  !
  !
LOOP3  A  1, DATA1
  !
  !
  END
```

```
MACRO
&ARG1  MACRO1  &COUNT, &ARG2
&ARG1  A  1, &ARG2
       AIF (&COUNT EQU 1).FINI
       A  2, &ARG2
       AIF (&COUNT EQU 2).FINI
       A  3, &ARG2

.FINI MEND
```

This belong to macro ← (.) definition only

```
START
LOOP1  MACRO1  3, DATA1
  !
LOOP2  MACRO1  2, DATA1
  !
  !
LOOP3  MACRO1  1, DATA
  !
  !
END
```

⑥ Specify the syntax of a macro definition
⑦ Difference between positional and keyword argument
④ Difference between AIF and AGO
⑤ Discuss with the example how conditional macro can be constructed.
⑥ Discuss with the example how to create macros.

Thrusday 15/09/22

```
MACRO
&ARG1  MACRO1  &ARG2, &ARG3
&ARG1    A  1, &ARG2
         A  2, &ARG3
         A  3, &ARG2
       MEND
```

role of pass-1 definition gets evaluated
role of pass-2 Encounter macro call

Argument list Array (ALA)
Macro definition table (MDT)
Macro Name Table (MNT)

ALA

Index	Arguments (8B)
0	'ARG1bbbb'
1	'ARG2bbbb'
2	'ARG2bbbb'

MDT

MDTC (Assuming) ← counter

index	MD (80 bytes)
15	&ARG1 MACRO1 &ARG2 ,&ARG3
18	.#0 A 1,#1
1A	A 2,#2
18	A 3,#1
19	MEND

MNT

MNTC counter

index	MacroName (8B)	MDT index
04	MACRO1	15

MACRO call
LOOP1 MACRO1 DATA1, DATA2

In pass-2 tables get updated.

ALA — one to one mapping

index	Arguments (8B)
0	LOOP1
1	DATA1
2	DATA2

MACRO call
LOOP1 MACRO1 DATA1, DATA2

=

LOOP1 A, 1, DATA1
A, 2, DATA2
A, 3, DATA1

Friday 16/09/22

Loaders

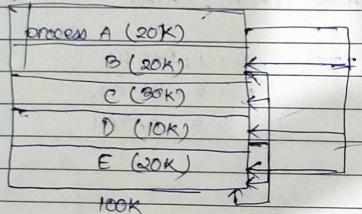
1. Compile and Go
2. General Loader scheme
3. Absolute Loader
4. Relocating Loader
5. Direct linking loaders

Concepts

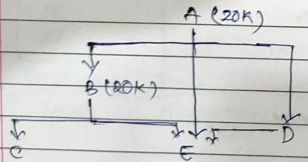
→ Dynamic loading
→ Overlay
→ Dynamic linking

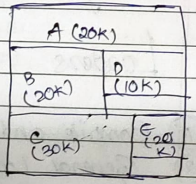
→ Allocation
→ Linking
→ Relocation
→ Loading

Overlay



memory size = 70K



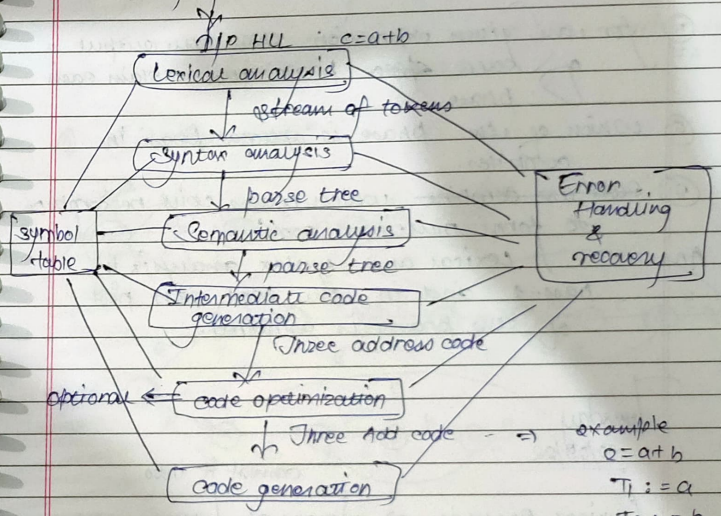


Q Write a R.E over the symbol a,b,c
starts with ab ends with ab & has a substring ab

(ab) (a+b+c)* (ab) (a+b+c)* (ab)

Monday 19/09/22

Phases of compiler



parse tree cannot generate without grammar.

Date / / Page

1. Difference between lexeme, token and pattern?
2. Difference between syntax and parse tree?
3. What is the need of lexical analysis
   syntax "
   ... "
4. Define compiler explain its phases in detail
5. for the given expression show the output of each ~~phase~~ phase and explain each phase.
6. which of the phase is optional phase in compiler.
7. ~~can we combine~~ which phases can be combine to form multipass compiler

Ans. lexical and syntax analysis in pass-1 and in the next pass rest of the phases is combined.

Tuesday
20/09/22

Lexical Analysis → stream of tokens (convert it into)

lexical analyser or scanner

Specification of token → Recognization of token

Date / / Page

→ for the specification of token regular expression is used

→ for recognization ⇒ finite automata

R.E ⟶ F.A/DFA
regular expression — finite automata.

R.E → NFA with ε → NFA without ε → DFA → minimized DFA

1. $a^*(bc)^*a(b+(cb)^*$