Apellido y Nombre CARBÓN POSSE SOFÍA Fecha: **18-02-2020**

email: ASOFI06@HOTMAIL.COM Cuatr. de cursada: 1C - 2019

Padrón: 101187 Turno de TP: ADIAS

Q

1) (a) Explicar qué es el ciclo de fetch y con qué finalidad se realiza.

(b) Explicar de qué manera se determina durante la ejecución de microcódigo la próxima microinstrucción a ser ejecutada.

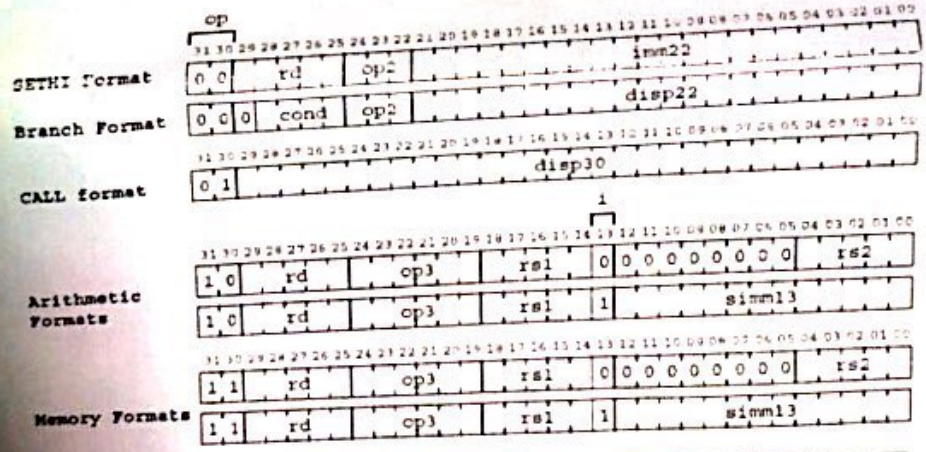
(c) Dar varios ejemplos de instrucciones de assembler cuya implementación en microcódigo requiere pasar por la microinstrucción almacenada en la posición 2047. Dar varios ejemplos de instrucciones que no requieren pasar por esa línea de microcódigo. Justificar su respuesta.

2) Un programa declara un arreglo de 32 palabras de 32 bits y carga en él las primeras 32 lecturas de un periférico que está mapeado en la dirección C3101200h. Una vez finalizada esta tarea invoca una subrutina que calcula su promedio. El programa principal devuelve el promedio por stack y termina.

Los valores entregados por el dispositivo están representados en un sistema numérico de complemento a 2.

La rutina, que debe ser declarada en el mismo módulo que el programa principal, recibe por stack la dirección del arreglo (el largo se siempre de 32 elementos) y devuelve también por stack el promedio calculado. En caso de excederse el rango de representación devuelve un cero.

3) Definir el contenido de la tabla de simbolos e indicar cuáles son las etapas en que su informacion es necesaria (considerando desde el compilado hasta el tiempo de ejecucion). Detallar en cada caso



op	Format
00	SETHI/Branch
01	CALL
10	Arithmetic
11	Memory

op2	Inst.
010	branch
100	sethi

op3 (op=10)	
010000	addcc
010001	andcc
010010	orcc
010110	orncc
100110	srl
111000	jmpl

op3 (op=11)	
000000	ld
000100	st

cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

F_3 F_2 F_1 F_0	Operation
0 0 0 0	ANDCC (A, B)
0 0 0 1	ORCC (A, B)
0 0 1 0	NORCC (A, B)
0 0 1 1	ADDCC (A, B)
0 1 0 0	SRL (A, B)
0 1 0 1	AND (A, B)
0 1 1 0	OR (A, B)
0 1 1 1	NOR (A, B)
1 0 0 0	ADD (A, B)
1 0 0 1	LSHIFT2 (A)
1 0 1 0	LSHIFT1C (A)
1 0 1 1	SIMM13 (A)
1 1 0 0	SEXT13 (A)
1 1 0 1	INC (A)
1 1 1 0	INCPC (A)
1 1 1 1	RSHIFTS (A)

cond	Operation
0 0 0	Use NEXT ADDR
0 0 1	Use JUMP ADDR if [31] = 1
0 1 0	Use JUMP ADDR if [30] = 1
0 1 1	Use JUMP ADDR if [29] = 1
1 0 0	Use JUMP ADDR if [28] = 1
1 0 1	Use JUMP ADDR if [27] = 1
1 1 0	Use JUMP ADDR if [26] = 1
1 1 1	DECODE

CARBÓN POSSE

FINAL ESTRUCTURA DEL COMPUTADOR
2° OPORTUNIDAD

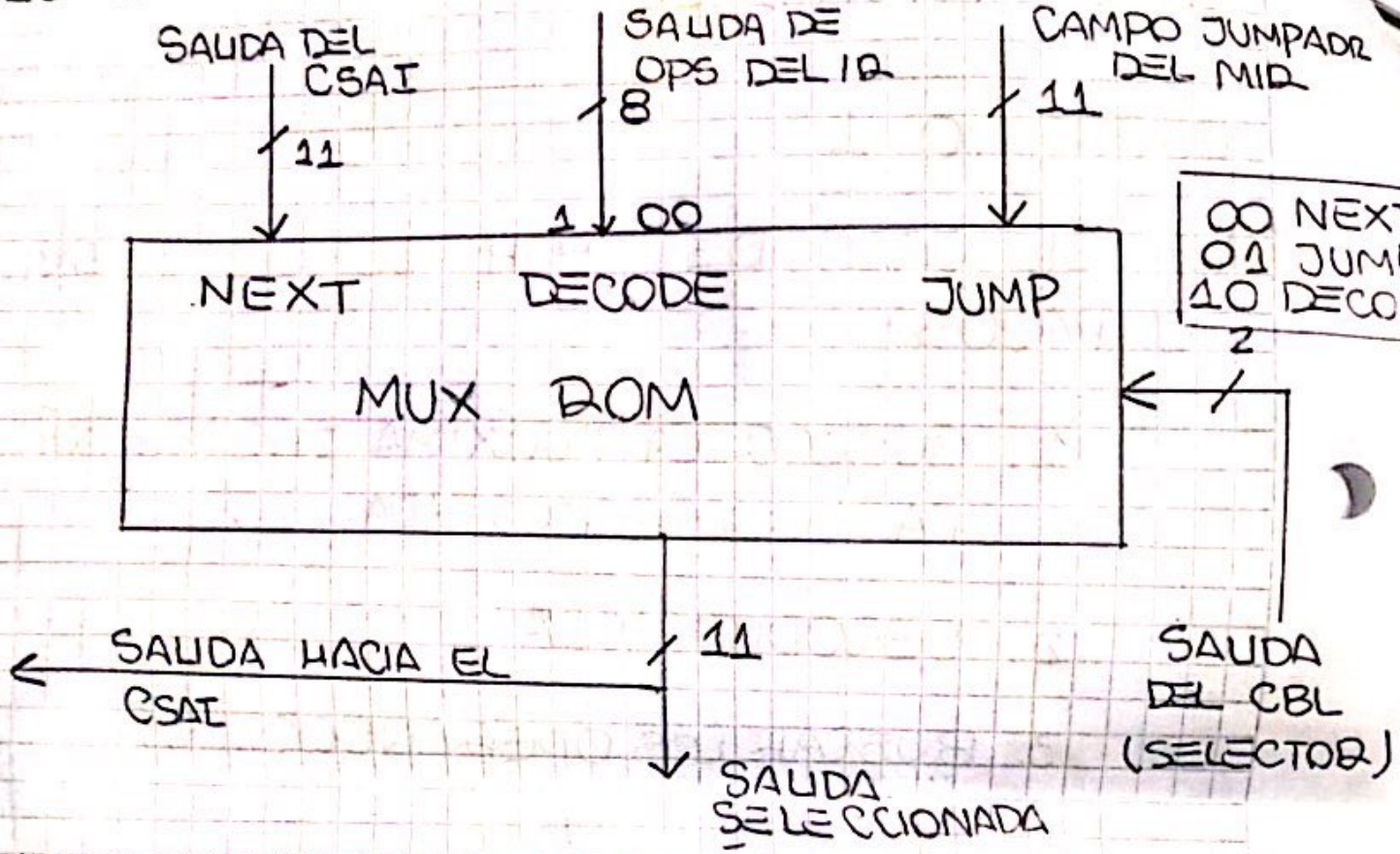
EJERCICIO 1

(A) EL CICLO DE FETCH CONSISTE EN 5 PASOS:

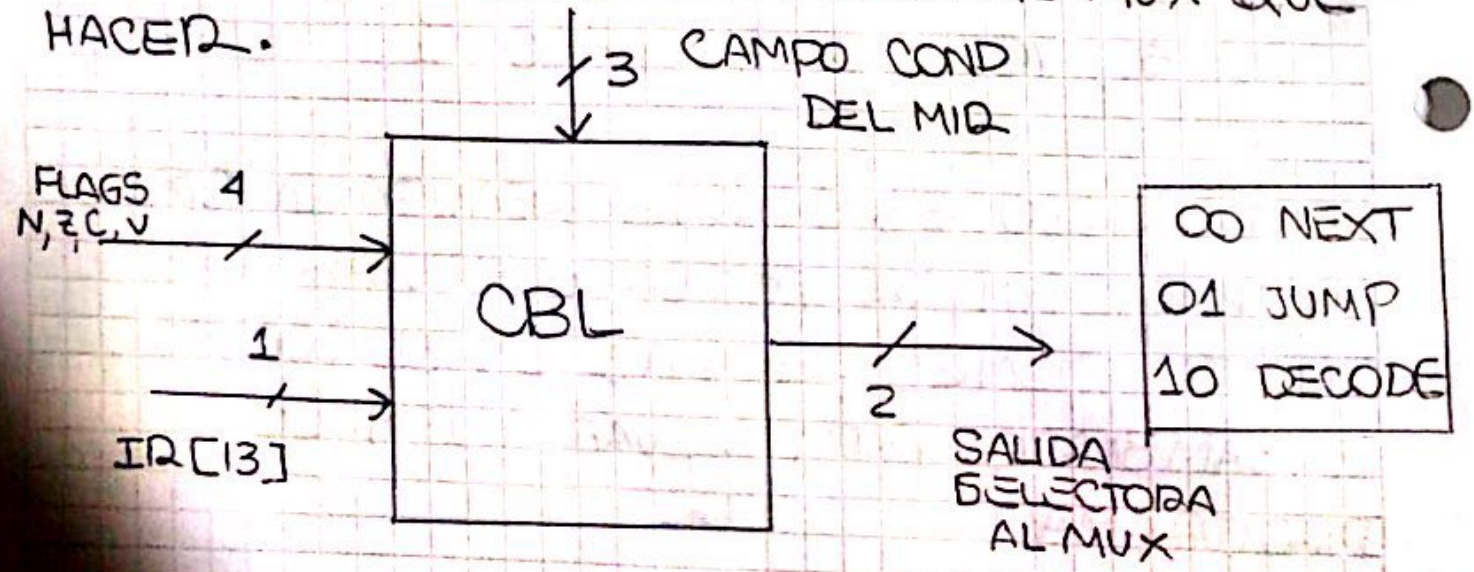
1. BUSCAR LA PRÓXIMA INSTRUCCIÓN A SER EJECUTADA.
2. DECODIFICARLA
3. BUSCAR LOS OPERANDOS SI ES QUE HAY.
4. EJECUTARLO
5. VOLVER AL PASO 1.

ESTE CICLO ES EL QUE REALIZA LA CPU CON EL FIN DE PODER EJECUTAR LAS INSTRUCCIONES PEDIDAS EN EL CODIGO ENSAMBLADO. PARA LLEVAR A FIN EL PROGRAMA.

(B) EL ENCARGADO DE SELECCIONAR LA PROXIMA MICROINSTRUCCIÓN A SER EJECUTADA ES EL MUX DE LA ROM. Y, JUNTO AL CBL.



EN REALIDAD LA SELECCION PROVIENE DEL 'CBL' QUE ESTE LE INDICA AL MUX QUE HACER.



CARBÓN POSSE

EL CBL RECIBE 3BITS DEL CAMPO 'COND' DEL MIR, ESTE CAMPO LE DICE AL CBL QUE HACER Y EL RESULTADO DE ESTA OPERACIÓN SE LA INDICA AL MUX DE LA ROM.

POR EJEMPLO, SI EL CAMPO COND DEL MIR CONTINE:

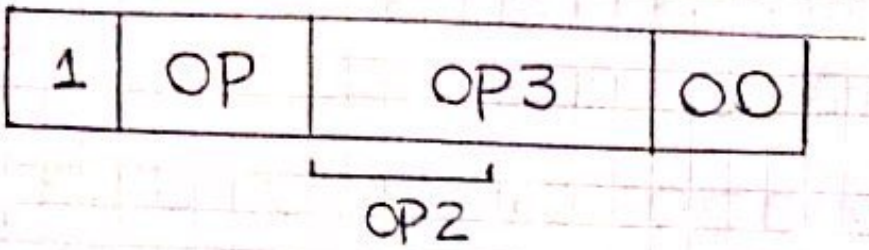
- 000: AUTOMATICAMENTE EL CBL LE ENVIA ↓ AL MUX QUE LA PROXIMA INSTRUCCIÓN A SER EJECUTADA ES 'NEXT', LA SIGUIENTE LINEA.

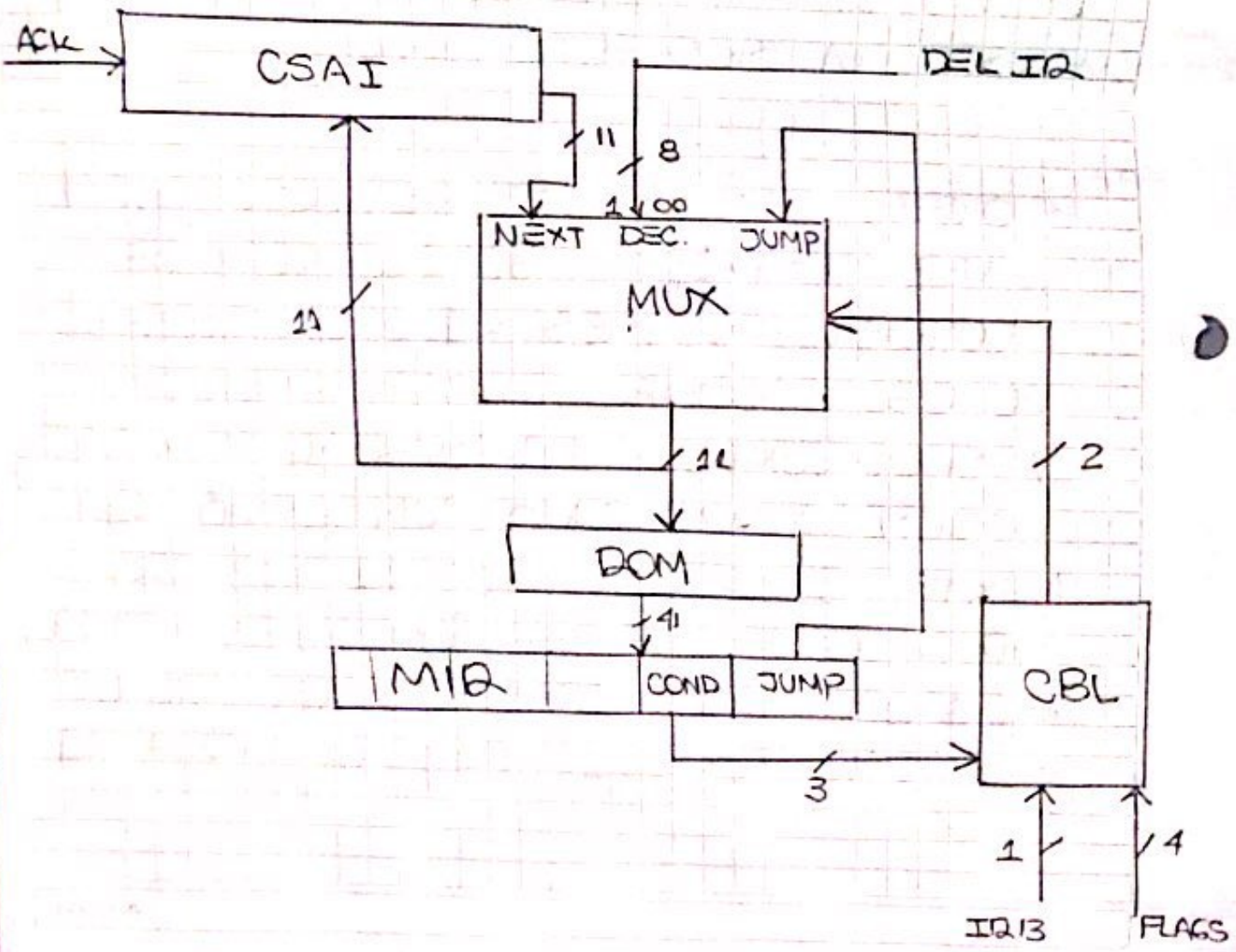
- 001: ESTOS BITS REPRESENTAN 'USE JUMPADDR' SI EL FLAG DE CONDICIÓN - N ES = 1. ENTONCES EL CBL COBROBORA ESTA CONDICIÓN Y SI LA CUMPLE LE MANDA AL MUX LA OPCIÓN 'JUMP' CASO CONTRARIO USA NEXT. JUMP ENVIA LOS 11 BITS DEL MIR QUE PUEDE SER CUALQUIER LINEA.

- 110: AUTOMATICAMENTE EL CBL LE ENVIA AL MUX QUE UTILIZE LA SALIDA DE JUMP.

POR ÚLTIMO,

- **111**: AUTOMÁTICAMENTE EL CBL LE ENVÍA AL MUX LA SALIDA DECODE. ESTA CONSISTE EN DECODIFICAR LA MICROINSTRUCCIÓN EN EL MIR, PARA VER A QUE LÍNEA SALTO.





CARBONÍ POSSE

(C)

MICROCODIGO QUE REQUIERE PASAR POR LA MICRO INSTRUCCIÓN EN LA POSICIÓN 2047:

EJEMPLO 1 - ADDCC

0: R[IR] ← AND (R[PC], R[PC]); READ
1: DECODE (1100 1000 0000)

1600: IF R[IR[13]] GO TO 1602;
1601: R[RD] ← ADDCC (R[RS1], R[RS2]);
GO TO 2047.
1602: R[TEMP0] ← SEXT13 [R[IR]]
1603: R[RD] ← ADDCC (R[RS1], R[TP0])
GO TO 2047;

2047: R[PC] ← INCPC [PC];
GO TO 0.

TMB SE UTILIZA DE LA MISMA FORMA EN ANDCC, NORCC, ORCC.

EJEMPLO 2 - LOAD

0: R[IR] ← AND (R[PC], R[PC]); READ
1: DECODE

1792: R[TEMP0] ← ADD (RS1, RS2)
IF R[IR[13]] THEN GO TO 1794;

1793: R[RD] ← AND (R[TEMP0], R[TEMP0])
GO TO 2047; READ

1794: R[TEMP0] ← SEXT13 [R[IR]]

1795: R[TEMP0] ← ADD (R[RS1], R[TEMP0])
GO TO 1793;

ESTAS INSTRUCCIONES NECESITAN PASAR POR LA LINEA 2047 PORQUE LUEGO DE EJECUTARLA, SE NECESITA PASAR A LA PROXIMA INSTRUCCIÓN DEL CODIGO QUE JUSTAMENTE ES LO QUE HACE ESA LINEA LE INCREMENTA 4 AL CONTADOR. (SON INSTRUCCIONES DE 32 BITS)

AHORA EN LA PROXIMA HOJA VAN LOS EJEMPLOS DE ~~QU~~ CUALES NO LA NECESITAN.

CARBÓN POSSE

EJEMPLO 1 - CALL

0: R[IR] ← AND (R[PC], R[PC]), READ
1: DECODE (SE COMPLETA ARBITRARIAMENTE)

1280: R[15] ← AND (R[PC], R[PC])
1281: R[TEMP0] ← ADD (R[IR], R[IR])
1282: R[TEMP0] ← ADD (R[TEMP0], R[T0])
1283: R[PC] ← ADD [R[PC], R[TEMP0])
GO TO 0;

EJEMPLO 2 - JUMPL

MISMAS LINEAS 0: Y 1:

1760: IF R[IR[13]) GO TO 1762.
1761: R[PC] ← ADD (R[RS1], R[RS2])
GO TO 0.
1762: R[TEMP0] ← SEXT13 (R[IR])
1763: R[PC] ← ADD (R[RSΔ], TEMP0)
GO TO 0;

ESTAS DOS MICROINSTRUCCIONES NO UTILIZAN LA LINEA 2047 PORQUE SU FUNCION ES INCREMENTAR PC O DISINCREMENTAR PC CON LA IDEA QUE LEA LO QUE HAY AHÍ (SUBRUTINAS) SI PASA POR 2047 INCREMENTARIA EN 4 Y SU FUNCION SERIA INCORRECTA.

EJERCICIO 2

```
• BEGIN
.• ORG 2048
%SP .EQU %R14
ADDCC %R15, %R0, %R16
COMIENZO_ARREGLO .dWb 32
SETHI C3101h, %R1
SLL %R1, 2, %R1
ADDCC %R1, 200h, %R1
ADDCC %R0, 128, %R2     ! R2 LARGO DEL ARRAY.
ADDCC %R0, %R0, %R3     ! R3 ES UN AUX = 0

LOOP: ANDCC %R2, %R2, %R0
      BE FIN
      ADDCC %R2, -4, %R2
      LD %R1, %R3
      ADDCC %R2, %R5, %R4
      ST %R3, %R4       ! R4 POSICION ACTUAL
                          DEL ARRAY
      BA LOOP
FIN : ADDCC %SP, -4, %SP
      ST %R5, %SP
      CALL PROMEDIO
      LD %SP, %R18
      JUMPL %R16+4, %R0
```

CARGO EN R5 EL COMIENZO DEL ARRAY

ADDCC %R0, 2052, R5

! R1 CONTIENE LA DIREC DEL PERIF.

CABRÓN POSSE

```
PROMEDIO : LD %SP, %R5
           ADDCC %SP, 4, %SP
LOOP-2:ANDCC %R2, %R2, %R0
           BE FIN
           ADDCC %R2, -4, %R2
           ADDCC %R0, %R0, %R6
           LD %R5, %R2, %R7
           ADDCC %R6, %R7, %R6
           BVS DESBORDE
           BA LOOP_2
FIN: SRL %R6, 5, %R10
           ADDCC SP, -4, SP
           ST %R10, %SP
           JUMPL %R15+4, %R0

DESBORDE : ADDCC %SP, -4, %SP
           ST %R0, %SP
           JUMPL %R15+4, %R0
```

[illegible]

```
. END
```

EJERCICIO 3

LA COMPILACIÓN ES EL PROCESO DE PASAR UN CÓDIGO DE ALTO NIVEL A ASSEMBLER. LUEGO EL ~~PROCESO DE~~ ENSAMBLADO ES EL PROCESO DE PASAR DEL CÓDIGO ENSAMBLADO A CÓDIGO DE MAQUINA.
EL ENSAMBLADOR DEL ABC ES UNO DEL TIPO DE DOS PASADAS. ESTO QUIERE DECIR QUE LEE EL CÓDIGO DOS VECES.

EN LA PRIMER PASADA, EL ENSAMBLADOR GENERA UNA TABLA DE SÍMBOLOS CON TODOS LOS RÓTULOS Y SUBRUTINAS CON SUS RESPECTIVOS VALORES. ADEMÁS IDENTIFICA SI ESTOS RÓTULOS SON .EXTERN O .GLOBALS, ESTO ES PARA IR A OTRO MÓDULO (EN EL CASO DE .EXTERN) A BUSCAR SU CONTENIDO PARA PODER ENSAMBLARLO Y EN EL CASO DE .GLOBAL PARA SABER QUE PUEDE SER UTILIZADO EN OTRO MÓDULO. DE ESTA SOLUCIÓN SE ENCARGA EL LINKER LUEGO, LA TABLA SOLO LE INDICA.
POR ÚLTIMO INDICA SI ESTOS SIMBOLOS SON REUBICABLES O NO. ESTO ES PARA QUE EL LINKER, A LA HORA DE ENLAZAR MÓDULOS SI LLEGA A VER UN PROBLEMA DE PISADA. (POR EJEMPLO DÓS MODULOS COMIENZAN CON .ORG 2048) PUEDA SABER SI SON REUBICABLES O NO.

6/6

CABBOÑ DOSSE

LA VENTAJA DE CREAR ESTA TABLA DE SIMBOLOS ES QUE PERMITE UTILIZAR UN RÓTULO ANTES DE DECLARARLO.

LUEGO SE REALIZA LA SEGUNDA PASADA, QUE GENERA EL CODIGO DE MAQUINA, UTILIZANDO EL CONTENIDO DE LA TABLA. POR ÚLTIMO SE ENLAZA Y SE CARGA.

UN EJEMPLO DE TABLA.

SIMBOLO	VALOR	G/E	REUBICABLE
LOOP	2062	—	Si
MAIN	3000	G	Si

. leaer?

2