Circuitos Digitales II y Laboratorio
Electrónica Digital II y Laboratorio

Fundamentos de Arquitectura de Computadores
Modelo de von Neumann

Profesor: Felipe Cabarcas

Correo:cabarcas@udea.edu.co

Oficina: 19-446

Departamento de Ingeniería Electrónica

Universidad de Antioquia

2013-1

El ISA

- El ISA es la interfase entre los comandos del SW y lo que realiza el HW.
- Empezaremos estudiando el lenguaje de máquina (no assembler)
- El ISA especifica toda la información sobre el computador, que el software tiene que saber.
- Todo lo que está disponible para un programador de lenguaje de máquina
- Se puede decir que el ISA especifica lo que está disponible para traducir un programa de alto nivel a lenguaje de máquina.
- El ISA especifica: la organización de la memoria, el conjunto de registros, el conjunto de instrucciones (incluyendo los opcode, tipos de datos y modos de direccionamiento)

La organización de la memoria

- El LC-3 tiene un espacio de direcciones de 2^{16} posiciones y una direccionabilidad de 16 bits.
- No todas las 65536 posiciones son realmente direcciones de memoria.

Registros

- Como la latencia de memoria es de más de 1 ciclo, la mayoría de los computadores tienen posiciones de almacenamiento con latencia 1.
- El tipo más común de este tipo de almacenamiento son los registros de propósito general (GPR).
- El LC-3 cuenta con 8 registros.
- Ejemplo: la instrucción 0001 010 000 0 00 001 – ADD R2 R0 R1 – R2 = R0 + R1

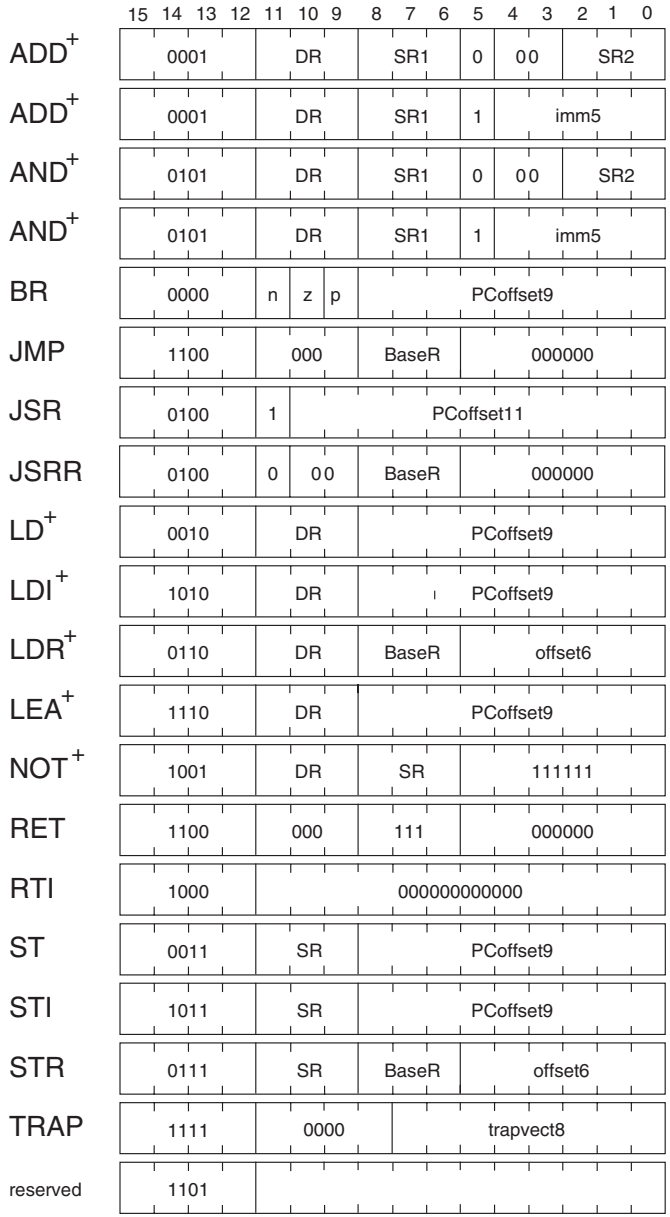
El conjunto de instrucciones

- Las instrucciones tiene dos partes:
- opcode, lo que debe hacer la instrucción
- operands, a quien le debe realizar la operación
- El conjunto de instrucciones está determinado por: los opcode, los tipos de datos, y los modos de direccionamiento.
- En el ejemplo del ADD, el modo de direccionamiento usado fue modo registro.
- La instrucción le solicita al computador que realice una suma en complemento a 2, entre registros de propósito general.

Opcodes

- Hay ISAs con pocas instrucciones otras con muchas ... ejemplos:
- multiply and add, MMX, Guardar el estado del procesador.
- El LC-3 tiene 15 instrucciones, y una (1101) reservada para "el futuro"
- Hay 3 tipos de instrucciones: Operaciones, movimiento de datos y control

LC-3 opcodes


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ADD+ 0001 DR SR1 0 00 SR2
ADD+ 0001 DR SR1 1 imm5
AND+ 0101 DR SR1 0 00 SR2
AND+ 0101 DR SR1 1 imm5
BR 0000 n z p PCoffset9
JMP 1100 000 BaseR 000000
JSR 0100 1 PCoffset11
JSRR 0100 0 00 BaseR 000000
LD+ 0010 DR PCoffset9
LDI+ 1010 DR PCoffset9
LDR+ 0110 DR BaseR offset6
LEA+ 1110 DR PCoffset9
NOT+ 1001 DR SR 111111
RET 1100 000 111 000000
RTI 1000 000000000000
ST 0011 SR PCoffset9
STI 1011 SR PCoffset9
STR 0111 SR BaseR offset6
TRAP 1111 0000 trapvect8
reserved 1101

Tipos de datos

- Es una representación de información tal que el ISA tenga opcodes que operen en esta representación.
- El LC-3 básicamente opera con enteros en complemento a 2, aunque la instrucción AND y la OR, operan en vector de bits.

Modos de direccionamiento

- Son los mecanismos para especificar donde están localizados los operandos.
- Generalmente se encuentran en:
- la memoria, en un registro, como parte de la instrucción.
- Si es parte de la instrucción, le llamamos literal o inmediato.
- El LC-3 soporta 5 modos:
- Inmediato, registro y 3 de memoria
- Relativo al PC, indirecto y base + desplazamiento (base + offset)

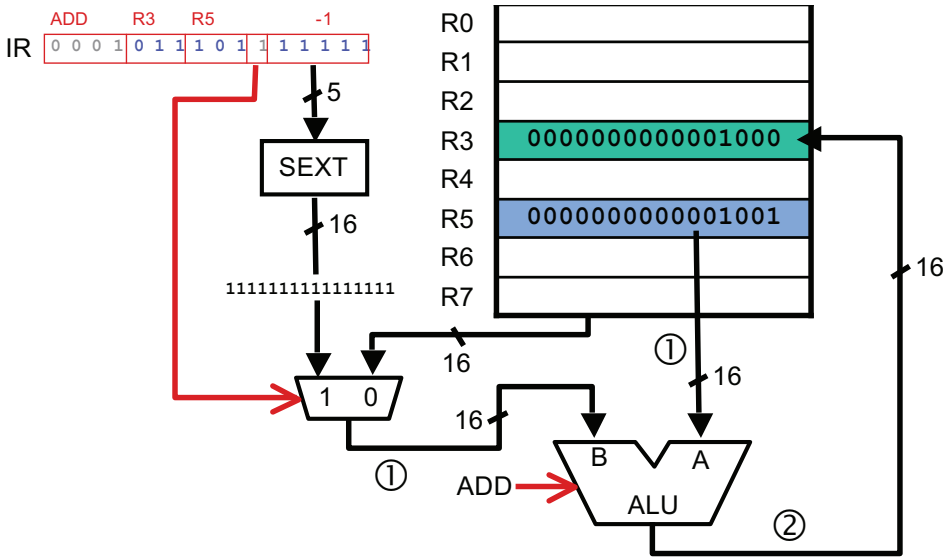
Códigos de condición

- Prácticamente todos los ISA permiten que la secuencia de instrucciones cambien dependiendo de resultados anteriores.
- El LC-3 tiene 3 registros de 1 bit que son escritos cada vez que uno de los 8 registros de propósito general son escritos.
- Se les llaman N, Z, y P, correspondiendo a Negativo, Cero, Positivo.
- Cada que se escribe uno de los registros, se escribe 0 o 1 en estos registros si el resultado es negativo, cero o positivo.

Instrucciones de operación

- Son operaciones aritméticas y lógicas
- El NOT (1001): niega los 16 bits. Es la única que usa un solo operando: usa modo de direccionamiento de registro para la fuente y el destino.
- El AND y AND tienen 2 operandos de 16 bits.
- AND (0101) hace operación "and" lógico con bit por bit.
- ADD (0001) hace suma en complemento a 2 de dos operandos de 16 bits.
- En AND y ADD uno de los operandos y el destino tienen direccionamiento de registro Bits[8:6] y Bits[11:9].
- El segundo operando fuente puede ser especificado por registro o por inmediato:
- si bit[5] es 0 entonces bits[2:0] son el registro fuente, y bits[4:3] deben ser 0
- si bit[5] es 1 entonces el operando son los bits[4:0] con signo extendido.
- ¿Qué enteros se pueden usar como inmediatos?

ADD, AND y NOT Data Path



Ejemplos

- ¿Qué hace 0101 011 010 1 00000? R3 = 0
- ¿Qué hace 0001 111 110 1 00001? R7 = R6 + 1
- Restar A y B (R0 y R1) y almacenarlo en R2
- 1001 001 001 111111 R1 = NOT(B) = NOT(R1)
- 0001 010 001 1 00001 R2 = -B – R1 + 1
- 0001 010 000 0 00 010 R2 = A + (-B) – R2 = R0 + R2
- ¿Qué resultado desastroso sucede en el código? ¿Cómo lo solucionamos?

Instrucción de movimiento de datos I

- Load: mover información de la memoria a los registros de propósito general
- store: mover información de los registros a la memoria.
- El LC-3 contiene 7 instrucciones: LD, LDR, LDI, LEA, ST, STR, y STI.
- Formato: opcode[15:12] DR o SR[11:9] Address Gen bits[8:0]
- Las instrucciones de movimiento de datos requiere 2 operandos: una fuente y un destino
- La fuente (source): es el dato a se movido
- El destino (destination) es la posición donde será movido.
- Uno será un registro y el segundo se asume que será la memoria (o dispositivo de entrada salida)
- Por ahora solo trabajaremos con memoria

Instrucción de movimiento de datos II

- Los bits[11:9] especifican el registro.
- Si la instrucción es load, DR es el registro destino.
- Si la instrucción es store, SR es el registro fuente.
- Los bits[8:0] contienen los bits para generación de direcciones (address generation bits)
- Estos bits son usados para calcular la dirección de 16 bits del segundo operando.
- En el LC-3 hay 4 maneras de interpretar estos bits: o modos de direccionamiento.
- El opcode determina la interpretación de estos bits.

Direccionamiento: Relativo al PC (PC-Relative)

- LD (0010) y ST (0011)
- Los bits[8:0] especifican un offset (desplazamiento) relativo al PC.
- La dirección de memoria es calculada extendiendo el signo de los 9 bits hasta 16 bits y sumándoselo al PC incrementado.
- La dirección solo puede estar entre +256 y -255 de la instrucción (¿por qué no -256 y 255?)
- ¿Cómo acceder a una dirección más lejana?

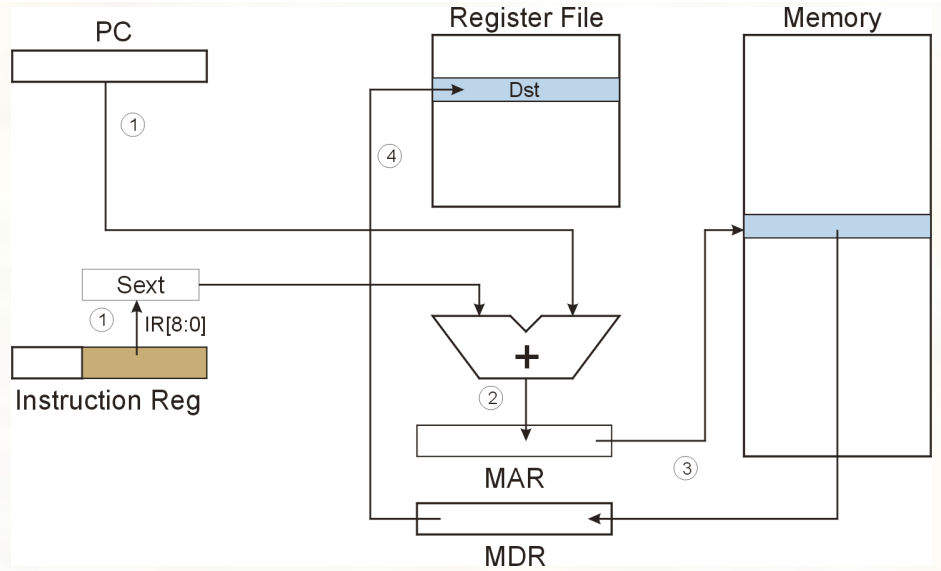


Figura: http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf

Direccionamiento: Indirecto

- LDI (1010) y STI (1011).
- Permite acceder a direcciones de memoria más allá de 256 posiciones de la instrucción: solución 1
- La dirección de memoria, calculada de la misma manera que en PC-relative, contiene la dirección donde se realizara el movimiento de datos.

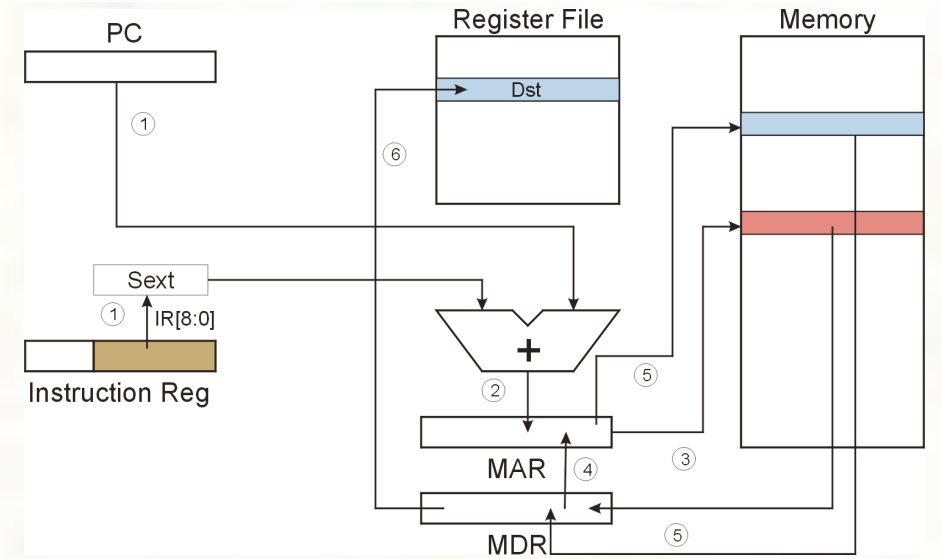


Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

Direccionamiento: Base + desplazamiento (Base+offset

- LDR (0110) y STR (0111).
- Permite acceder a direcciones de memoria más allá de 256 posiciones de la instrucción: solución 2
- La dirección es calculada sumando 6 bits (offset) extendido a un registro base
- Se le puede sumar -32 a +31 al registro

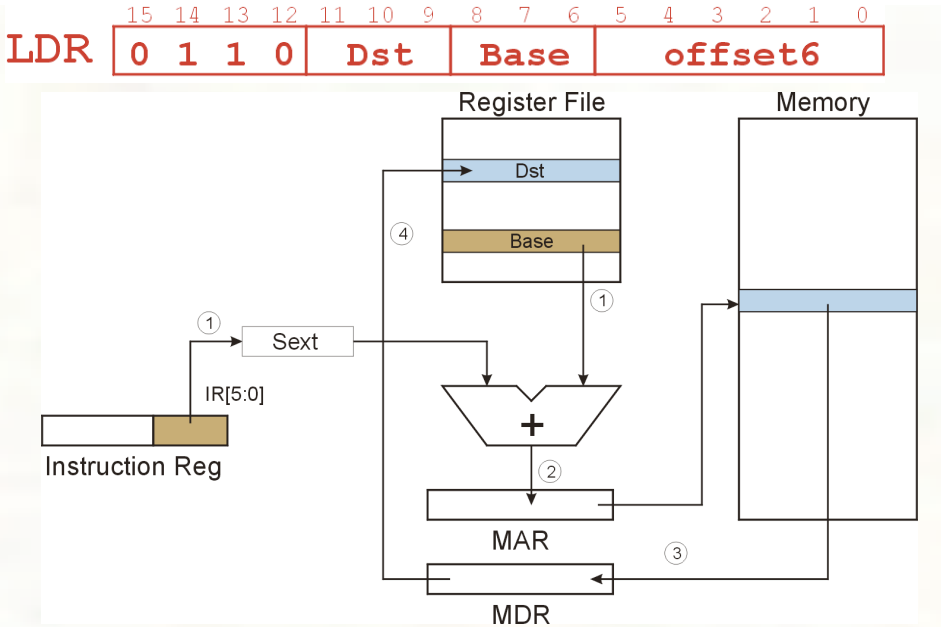


Figura: http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf

Direccionamiento: inmediato

- LEA (1110)
- Carga en un registro la el PC incrementado más los bits [8:0] con el signo extendido

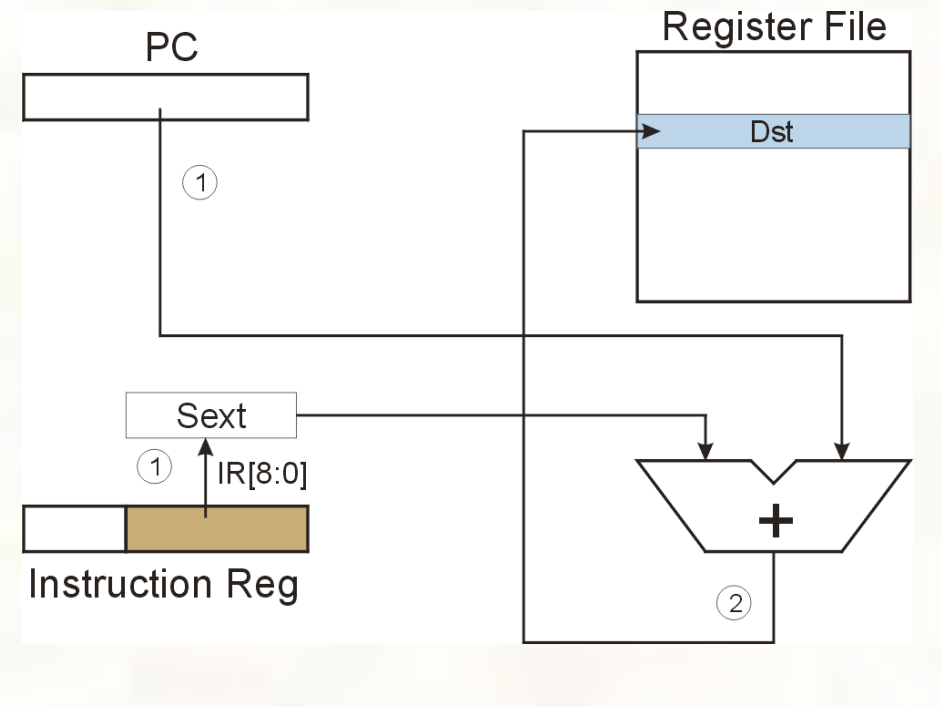


Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

Ejemplo

Address	*Instruction*	*Comments*
x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1	*R1 ← PC – 3 = x30F4*
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	*R2 ← R1 + 14 = x3102*
x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1	*M[PC - 5] ← R2* *M[x30F4] ← x3102*
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0*
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	*R2 ← R2 + 5 = 5*
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	*M[R1+14] ← R2* *M[x3102] ← 5*
x30FC	1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1	*R3 ← M[M[x30F4]]* *R3 ← M[x3102]* *R3 ← 5*
	opcode	

Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

Instrucciones de Control

- El LC-3 tiene 5 opcodes que permiten romper la secuencia de ejecución:
- salto condicional
- salto sin condición
- salto a subrutina (o a función)
- TRAP
- retorno de interrupción

Saltos condicionales

- Los registros N,Z,P se escriben cada vez que se escribe un registro (ADD, AND, NOT, LD, LDI, LDR, LEA): Solo uno de los bits es 1.

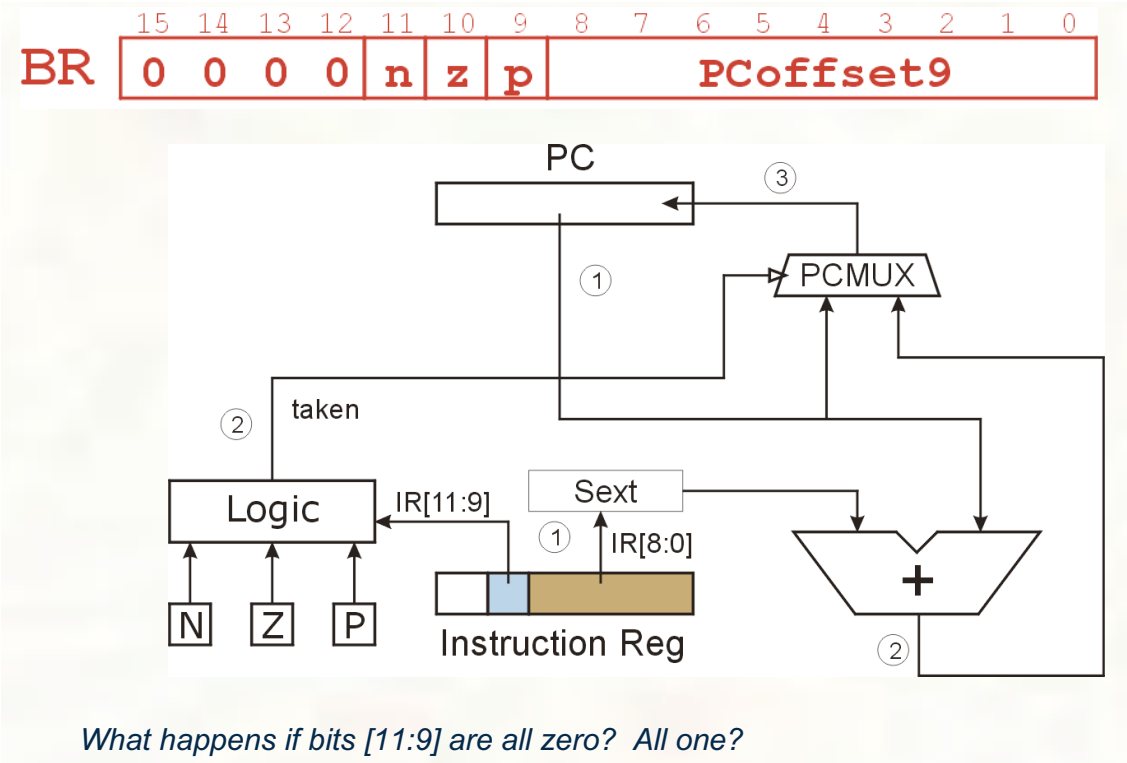
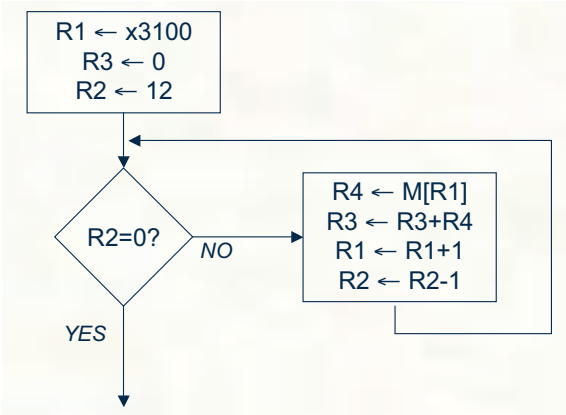


Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

Ejemplo

- Sumar 12 enteros que se encuentran en memoria a partir de la dirección x3100.
- El programa arranca en la dirección x3000.



Address	*Instruction*	*Comments*
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	*R1 ← x3100 (PC+0xFF)*
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ← 0*
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0*
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	*R2 ← 12*
x3004	0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1	*If Z, goto x300A (PC+5)*
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	*Load next value to R4*
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1	*Add to R3*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*Increment R1 (pointer)*
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	*Decrement R2 (counter)*
x3009	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0	*Goto x3004 (PC-6)*

Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

2 métodos para controlar bucles

- Se le llama bucle, o loop a una secuencia de instrucciones que se ejecutan repetidamente.
- Cada ejecución de las instrucciones es una iteración.
- 2 métodos comunes: usando un contador, con un centinela.
- El uso del contador es útil si sabemos el número de iteraciones.
- Cuando no sabemos el número de iteraciones, por ejemplo si queremos sumar una lista de números, en formato ASCII, podríamos usar un valor no numérico para representar ese último número del vector.
- Si en el ejemplo anterior no se sabe el número de elementos, pero se sabe que los números son positivos, se podría usar el -1 para parar la ejecución.

La instrucción JMP (salto incondicional)

- La instrucción BR, solo permite saltar a una instrucción que está +256 a -255 posiciones del PC.
- Para saltar a cualquier dirección, se puede usar JMP, que salta a una dirección almacenada en un registro

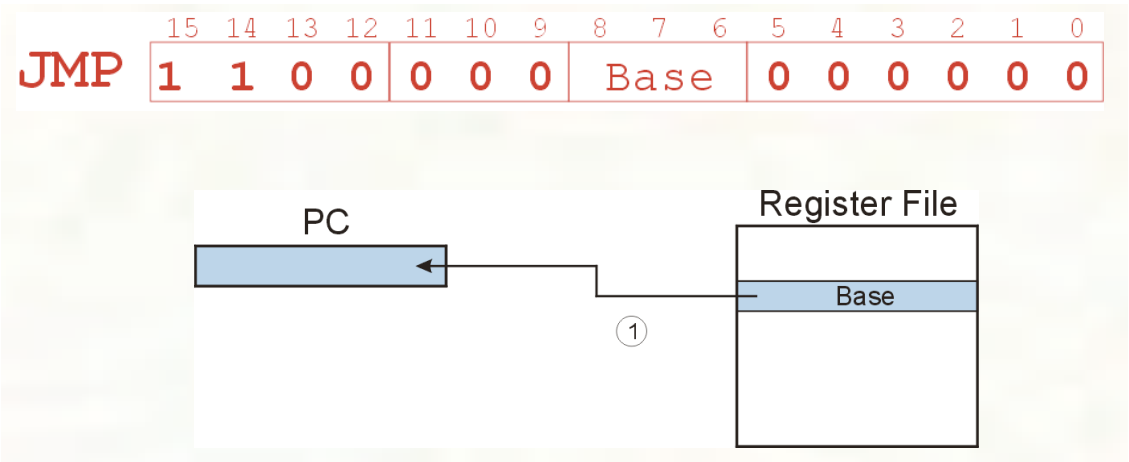


Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

La instrucción TRAP

- Formato: 1111 0000 trapvector[7:0]
- Cambia el PC a una dirección que es parte del sistema operativo
- El sistema operativo realiza alguna tarea para el programa que se está ejecutando
- El trapvector identifica el servicio, entre otros tenemos:
- x23: Entra un carácter desde el teclado
- x21: muestra un carácter en el monitor
- x25: Detiene el programa (Halt)

Ejemplo: Cuenta apariciones de carácter en archivo

- El programa inicia en x3000
- Leer carácter del teclado
- Cargar cada carácter del archivo
  - El archivo es una secuencia de posiciones de memoria
  - El comienzo del archivo está almacenado en la dirección siguiente al final del programa
- Si el carácter del archivo es igual al que se entra por teclado, se incrementa el contador
- El final del archivo está dado por el carácter especial: EOT (x04)
- Al final muestre el número de caracteres y suspenda ejecución (asuma que hay menos de 10 caracteres)

Algoritmo

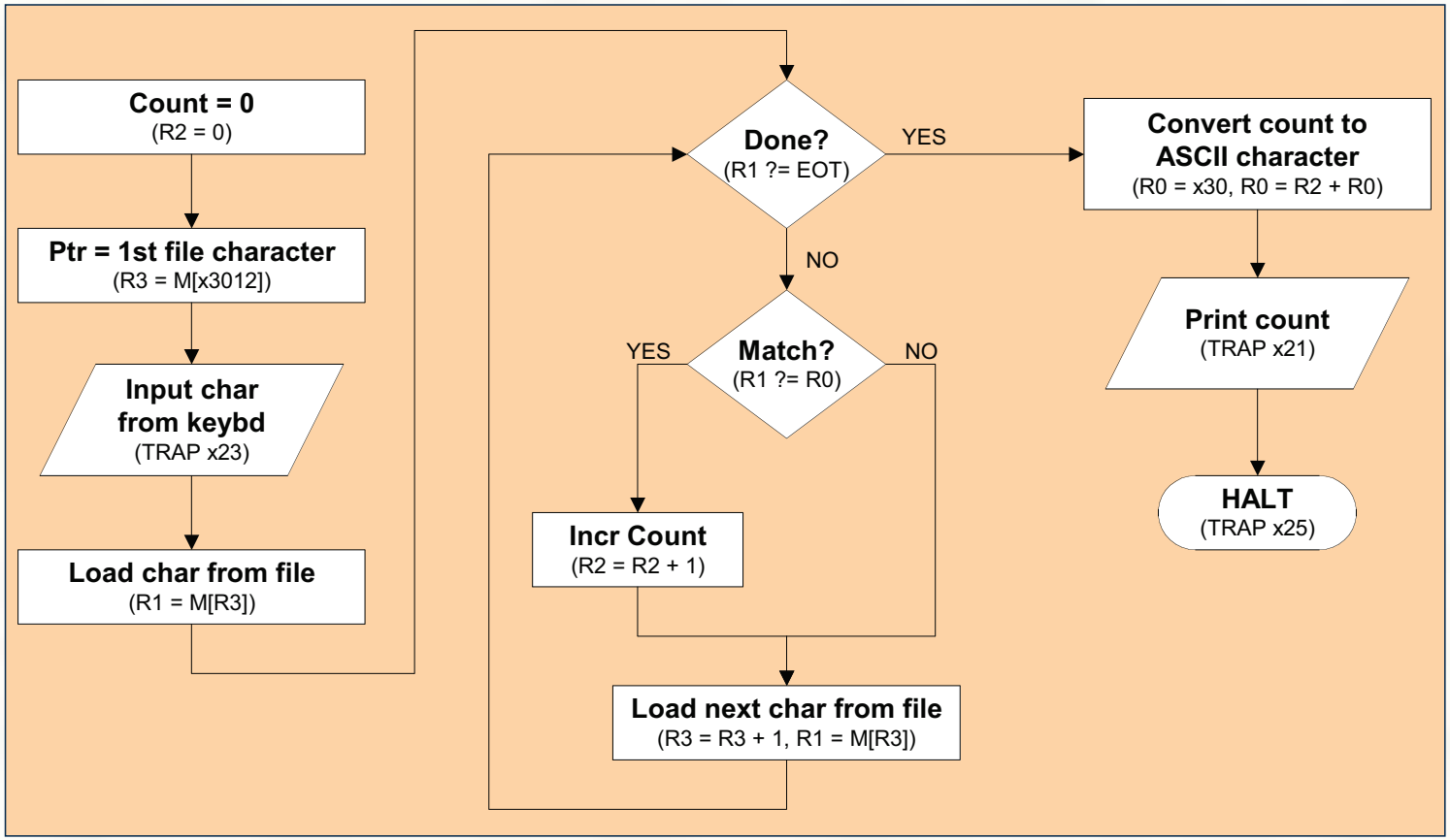


Figura: `http://www.cs.utexas.edu/users/fussell/courses/cs310h/lectures/Lecture_10-310h.pdf`

Programa

Address	*Instruction*	*Comments*
x3000	0101 010 010 1 00000	*R2 ← 0 (counter)*
x3001	0010 011 0000100000	*R3 ← M[x3012] (ptr)*
x3002	1111 0000 0010000011	*Input to R0 (TRAP x23)*
x3003	0110 001 011 000000	*R1 ← M[R3]*
x3004	0001 100 001 1 11100	*R4 ← R1 – 4 (EOT)*
x3005	0000 010 0000001000	*If Z, goto x300E*
x3006	1001 001 001 1 11111	*R1 ← NOT R1*
x3007	0001 001 001 1 00001	*R1 ← R1 + 1*
X3008	0001 001 001 0 00000	*R1 ← R1 + R0*
x3009	0000 101 0000000001	*If N or P, goto x300B*
x300A	0001 010 010 1 00001	*R2 ← R2 + 1*
x300B	0001 011 011 1 00001	*R3 ← R3 + 1*
x300C	0110 001 011 000000	*R1 ← M[R3]*
x300D	0000 111 1111101110	*Goto x3004*
x300E	0010 000 0000000100	*R0 ← M[x3013]*
x300F	0001 000 000 0 00010	*R0 ← R0 + R2*
x3010	1111 0000 0010000001	*Print R0 (TRAP x21)*
x3011	1111 0000 0010010101	*HALT (TRAP x25)*
X3012	Starting Address of File	
x3013	0000000000110000	*ASCII x30 ('0')*

LC3 datapath