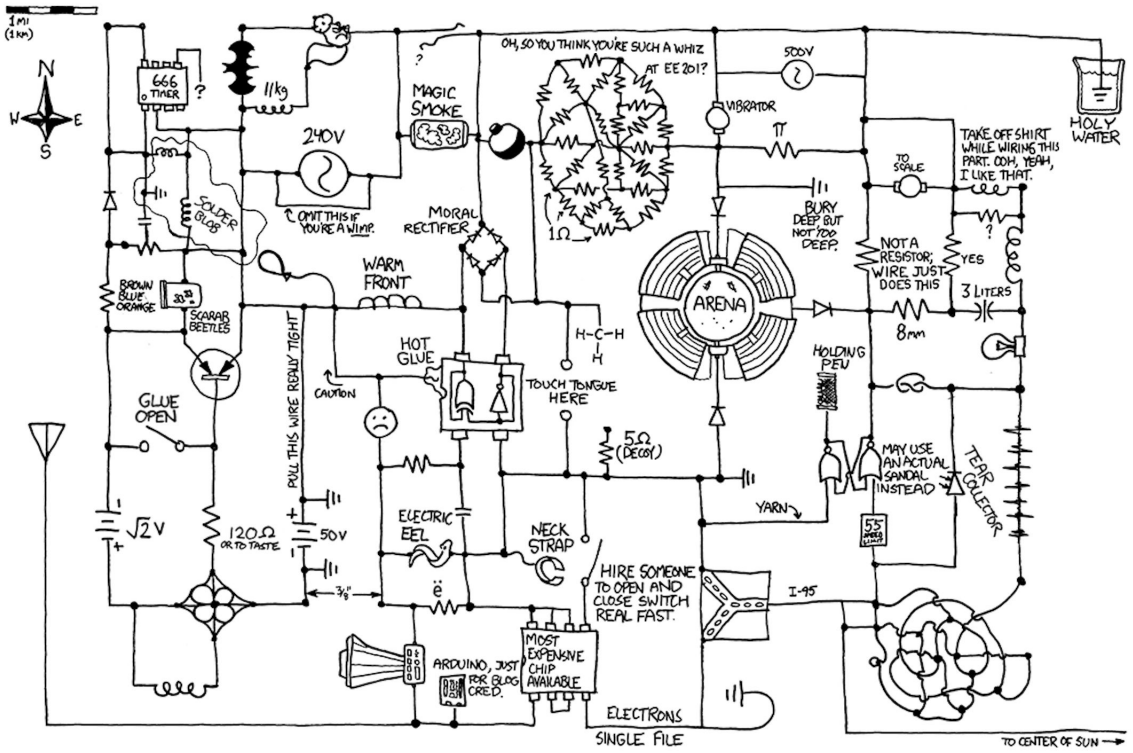
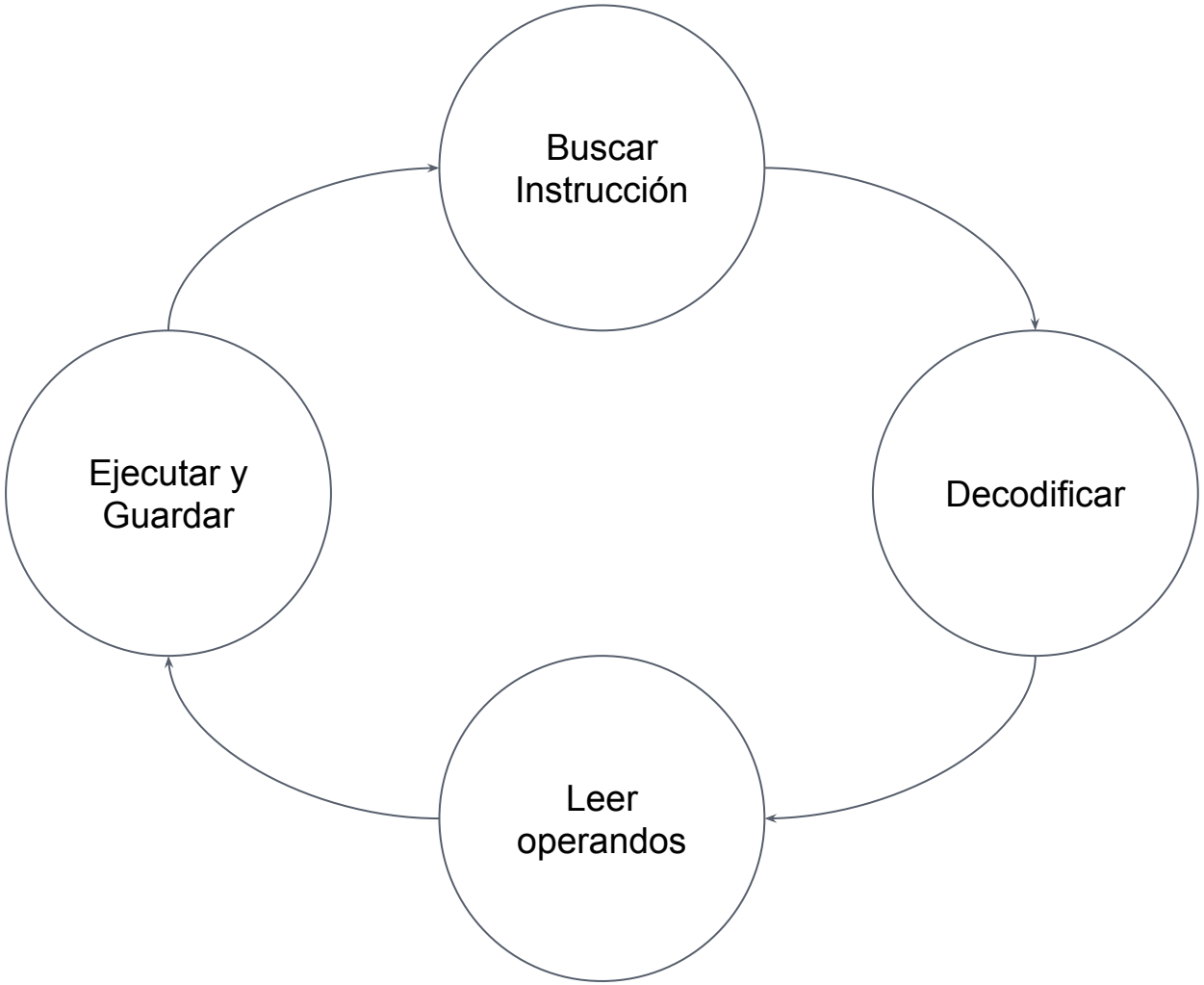
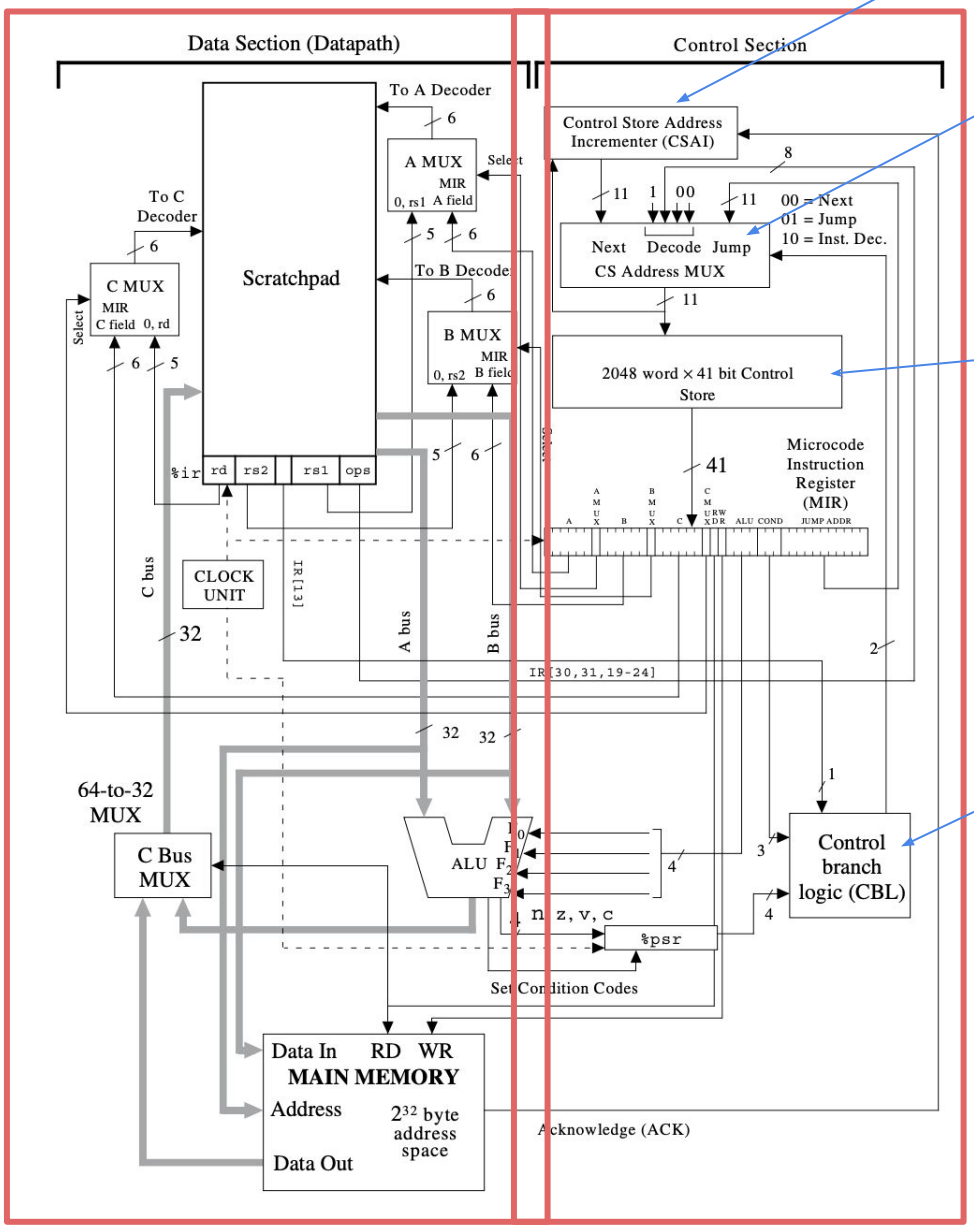
Microarquitecura



Ciclo de búsqueda



Introducción



Calcula la siguiente micro instrucción

Multiplexor para acceder a la memoria que almacena el micro código

Memoria que almacena el micro código

Lógica de control de saltos

Ejercicio

En un procesador ARC el registro Program Counter apunta a la siguiente instrucción guardada en RAM:

```
addcc  %r10, 48, %r1
```

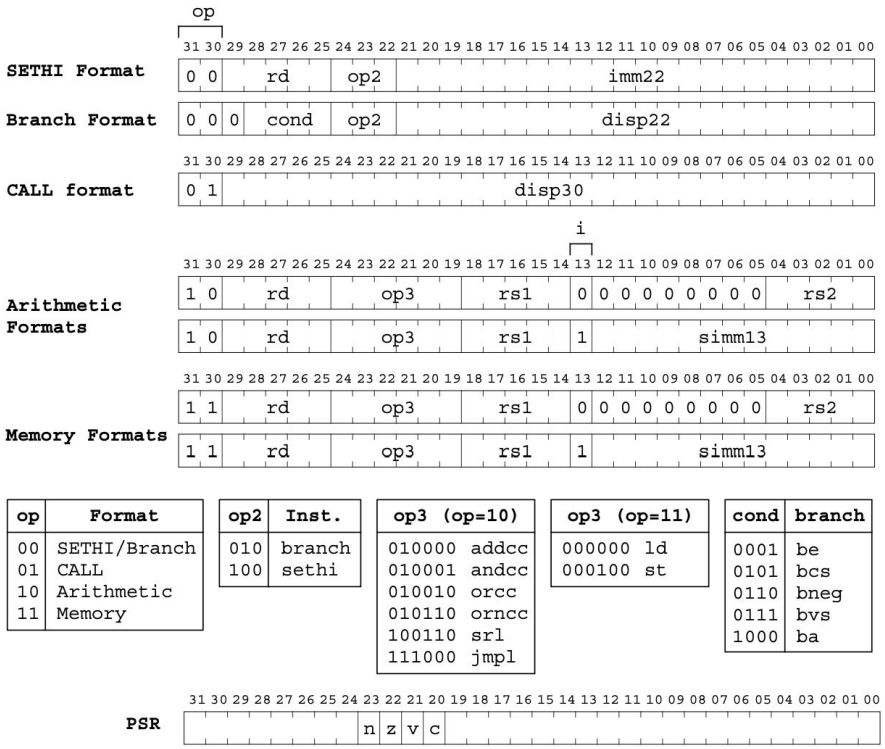
Detallar los pasos de microprograma que la decodifican y para cada uno de ellos indicar los valores presentes en las entradas y en las salidas de cada uno de los siguientes bloques funcionales:

- multiplexor de direcciones de la memoria de control
- incrementador de direcciones de la memoria de control
- decodificadores de los buses A B y C
- multiplexores que intervienen en la decodificación de los buses A B y C
- multiplexor de datos del bus C
- bus de datos A B y C
- entrada de direcciones del módulo de memoria RAM

Considere que antes de la ejecución de esta instrucción %r10 = %r1 = 0

Buscar Instrucción

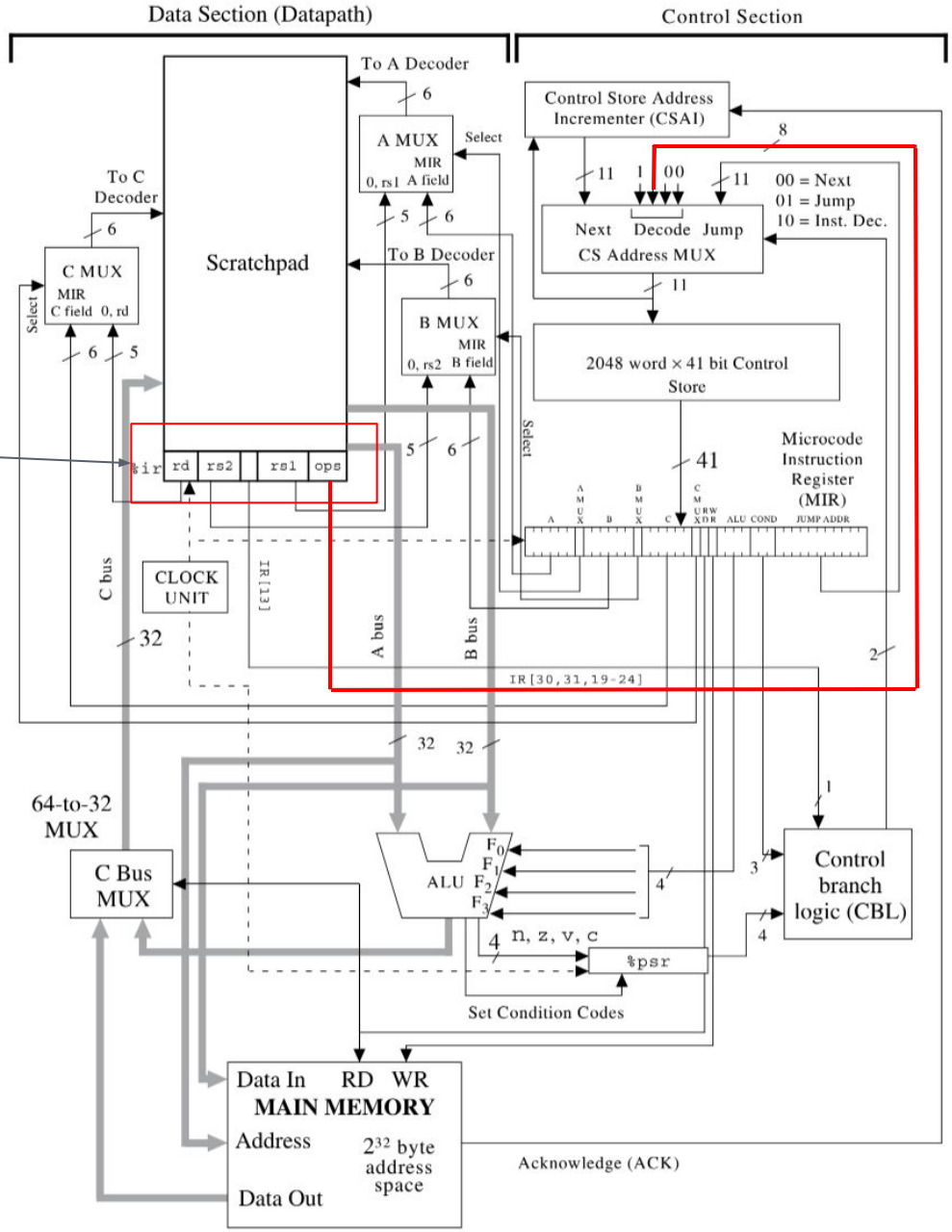
```
addcc %r10, 48, %r1
```



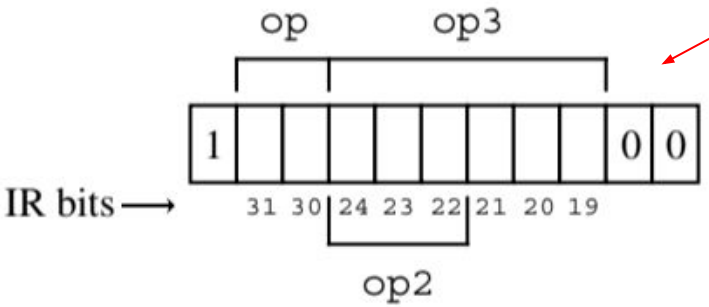
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0

Decodificar

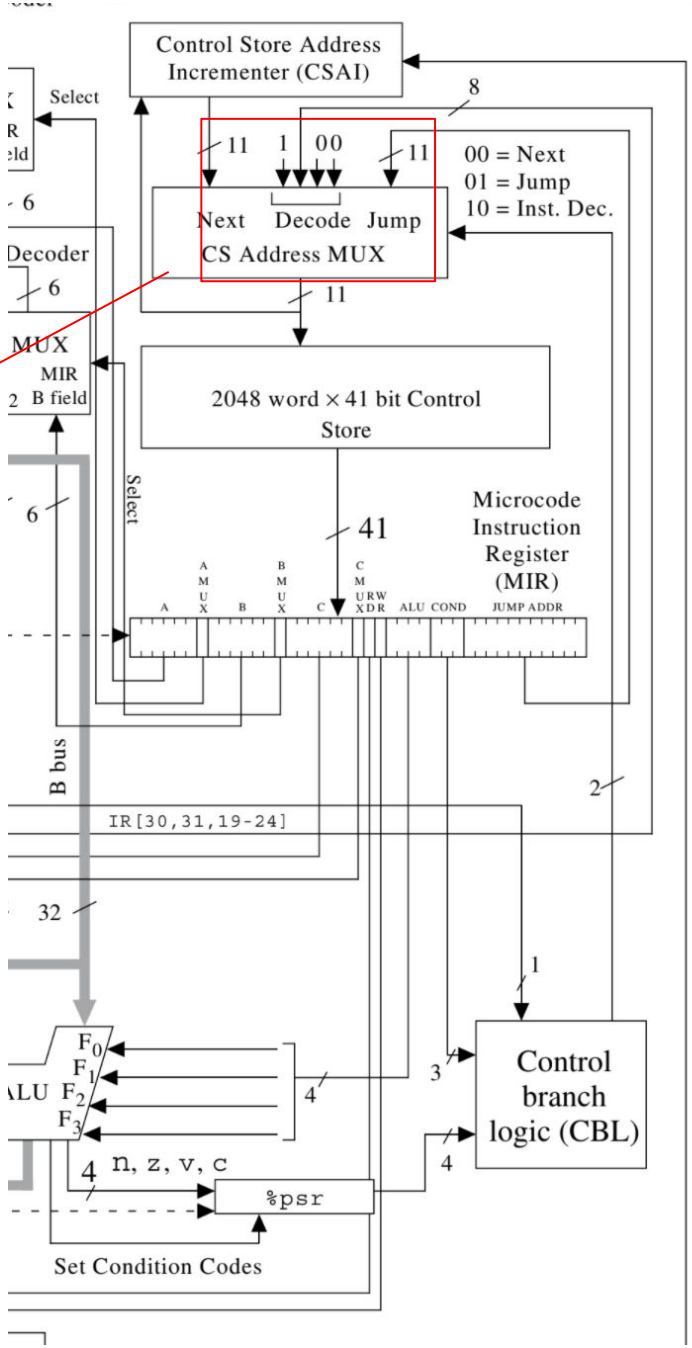
```
addcc %r10, 48, %r1
```



Decodificar



10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0	0	0	0



Aquí obtenemos la dirección de la microinstrucción a ejecutar 1600

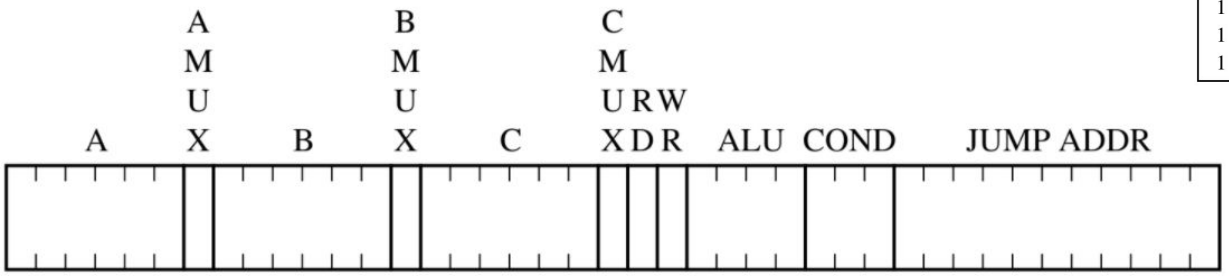
EJECUCIÓN

`addcc %r10, 48, %r1`

```
      / addcc
1600: IF R[IR[13]] THEN GOTO 1602;      / Is second source operand immediate?
1601: R[rd] ← ADDCC(R[rs1],R[rs2]);     / Perform ADDCC on register sources
      GOTO 2047;
1602: R[temp0] ← SEXT13(R[ir]);         / Get sign extended simm13 field
1603: R[rd] ← ADDCC(R[rs1],R[temp0]);   / Perform ADDCC on register/simm13
      GOTO 2047;                        / sources
```

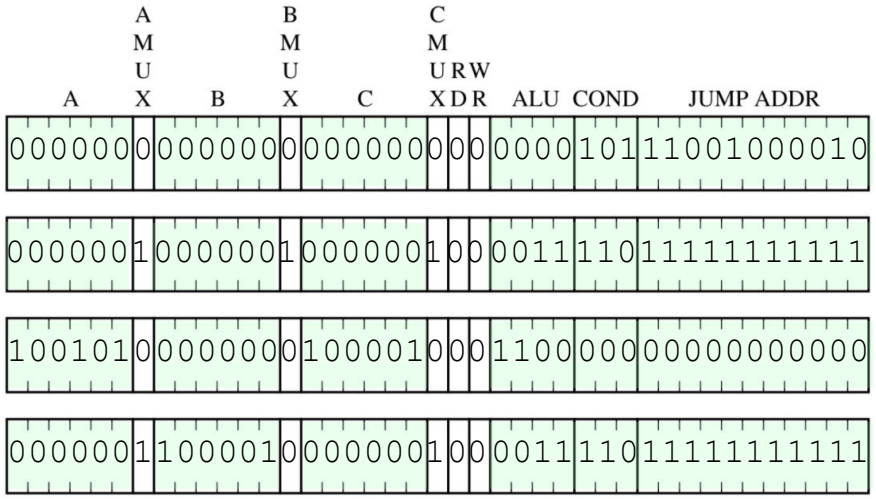
C_2	C_1	C_0	Operation
0	0	0	Use NEXT ADDR
0	0	1	Use JUMP ADDR if n = 1
0	1	0	Use JUMP ADDR if z = 1
0	1	1	Use JUMP ADDR if v = 1
1	0	0	Use JUMP ADDR if c = 1
1	0	1	Use JUMP ADDR if IR[13] = 1
1	1	0	Use JUMP ADDR
1	1	1	DECODE

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	LSHIFT10 (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	RSHIFT5 (A)



EJECUCIÓN

```
        / addcc
1600: IF R[IR[13]] THEN GOTO 1602;

1601: R[rd] ← ADDCC(R[rs1],R[rs2]);
      GOTO 2047;

1602: R[temp0] ← SEXT13(R[ir]);

1603: R[rd] ← ADDCC(R[rs1],R[temp0]);
      GOTO 2047;
```



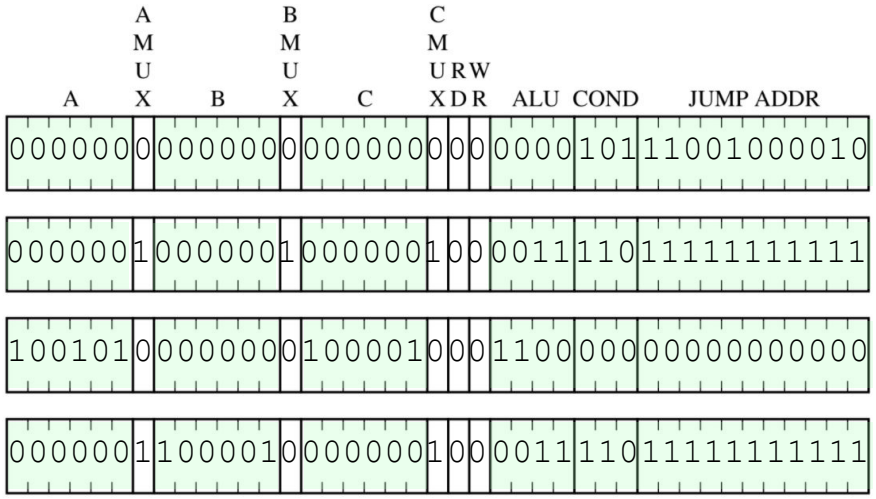
F_3 F_2 F_1 F_0	Operation
0 0 0 0	ANDCC (A, B)
0 0 0 1	ORCC (A, B)
0 0 1 0	NORCC (A, B)
0 0 1 1	ADDCC (A, B)
0 1 0 0	SRL (A, B)
0 1 0 1	AND (A, B)
0 1 1 0	OR (A, B)
0 1 1 1	NOR (A, B)
1 0 0 0	ADD (A, B)
1 0 0 1	LSHIFT2 (A)
1 0 1 0	LSHIFT10 (A)
1 0 1 1	SIMM13 (A)
1 1 0 0	SEXT13 (A)
1 1 0 1	INC (A)
1 1 1 0	INCPC (A)
1 1 1 1	RSHIFT5 (A)

C_2 C_1 C_0	Operation
0 0 0	Use NEXT ADDR
0 0 1	Use JUMP ADDR if n = 1
0 1 0	Use JUMP ADDR if z = 1
0 1 1	Use JUMP ADDR if v = 1
1 0 0	Use JUMP ADDR if c = 1
1 0 1	Use JUMP ADDR if IR[13] = 1
1 1 0	Use JUMP ADDR
1 1 1	DECODE

DECODIFICAR

addcc %r10, 48, %r1

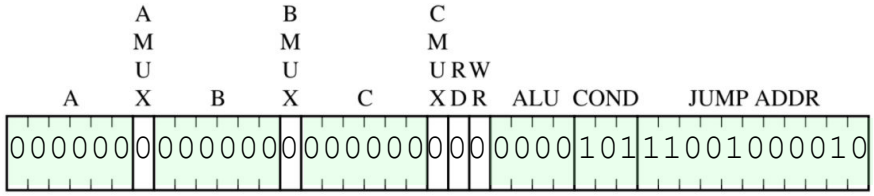
```
1600: IF R[IR[13]] THEN GOTO 1602;

1601: R[rd] ← ADDCC(R[rs1],R[rs2]);
      GOTO 2047;

1602: R[temp0] ← SEXT13(R[ir]);

1603: R[rd] ← ADDCC(R[rs1],R[temp0]);
      GOTO 2047;
```

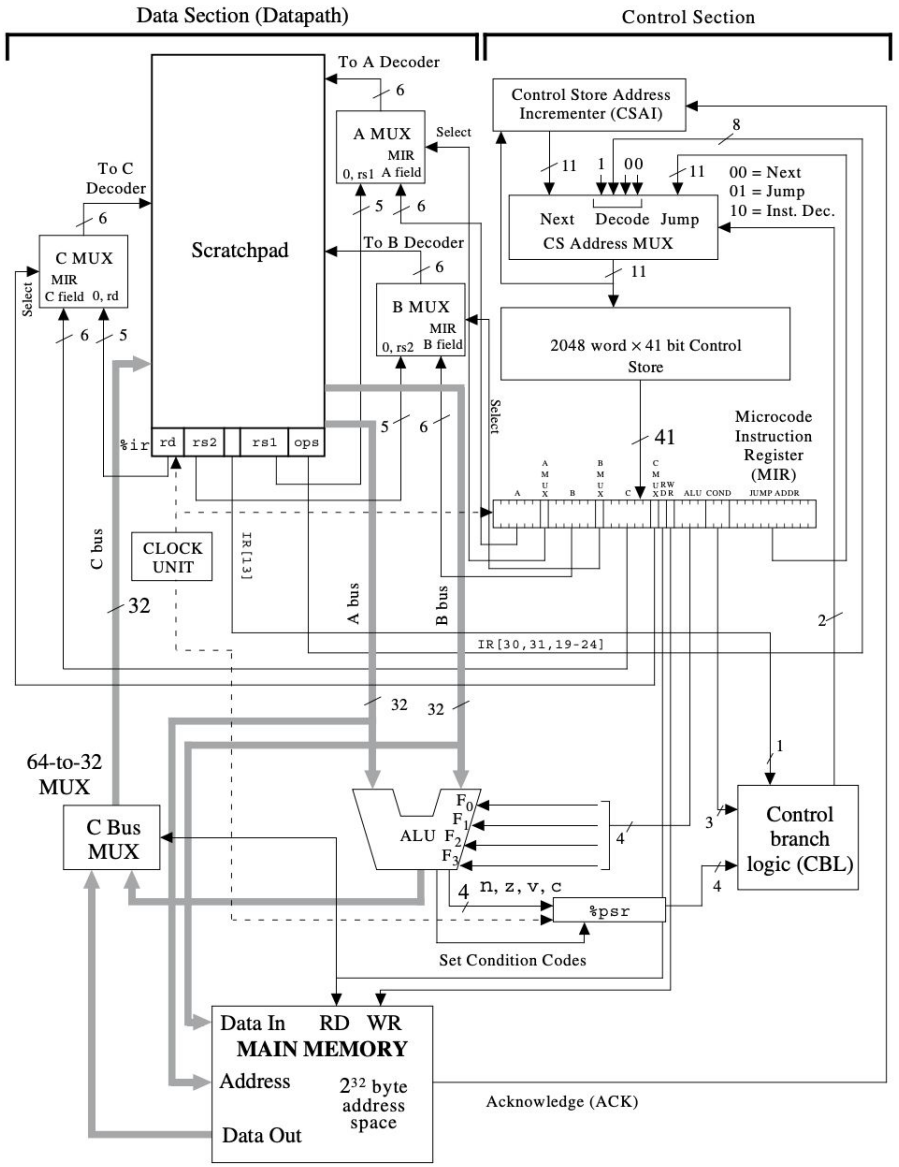


EJECUCIÓN

```
addcc %r10, 48, %r1
```

```
1600: IF R[IR[13]] THEN GOTO 1602;
```

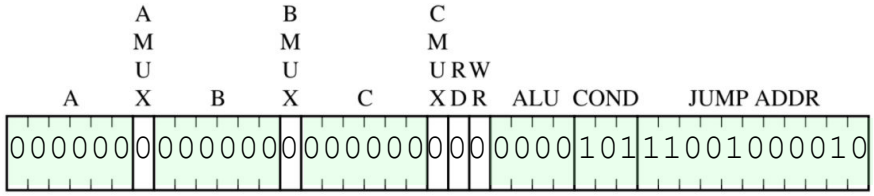


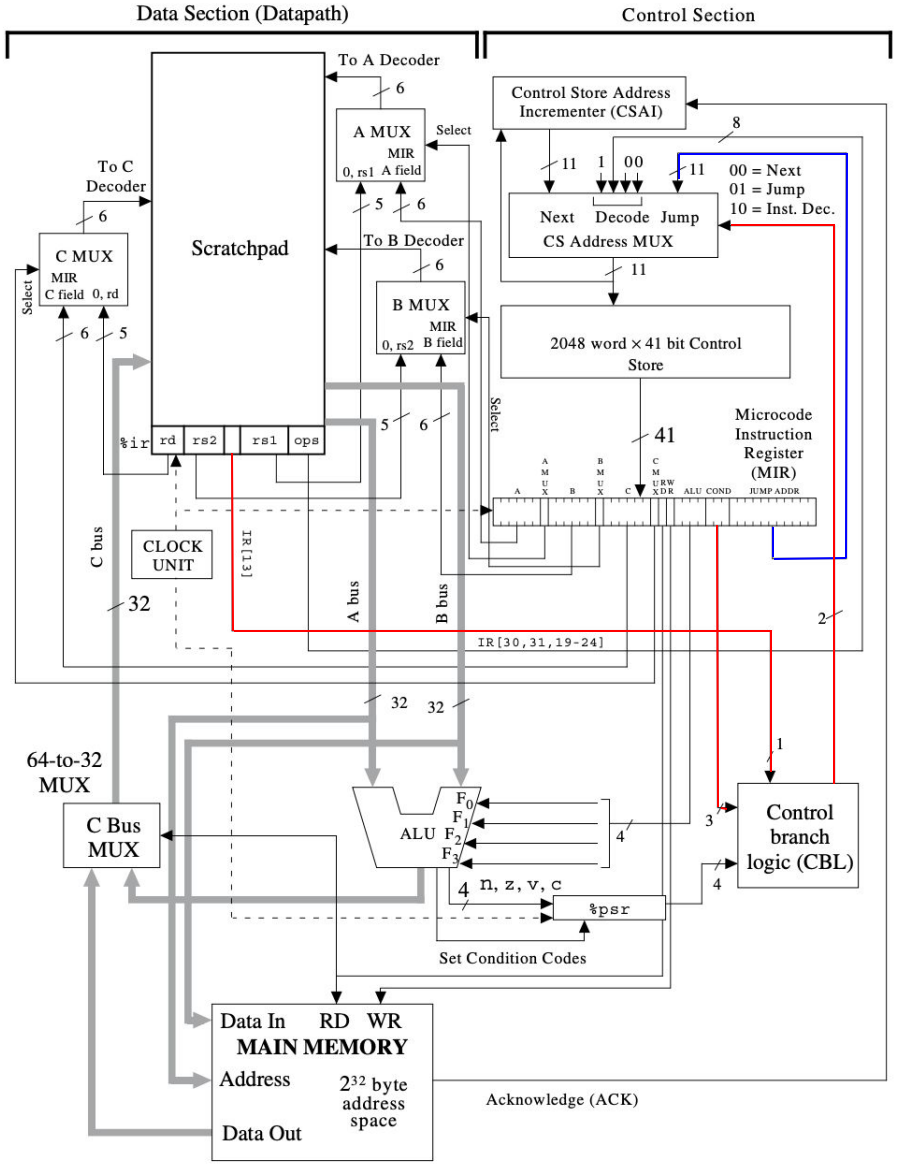


EJECUCIÓN

```
addcc %r10, 48, %r1
```

```
1600: IF R[IR[13]] THEN GOTO 1602;
```



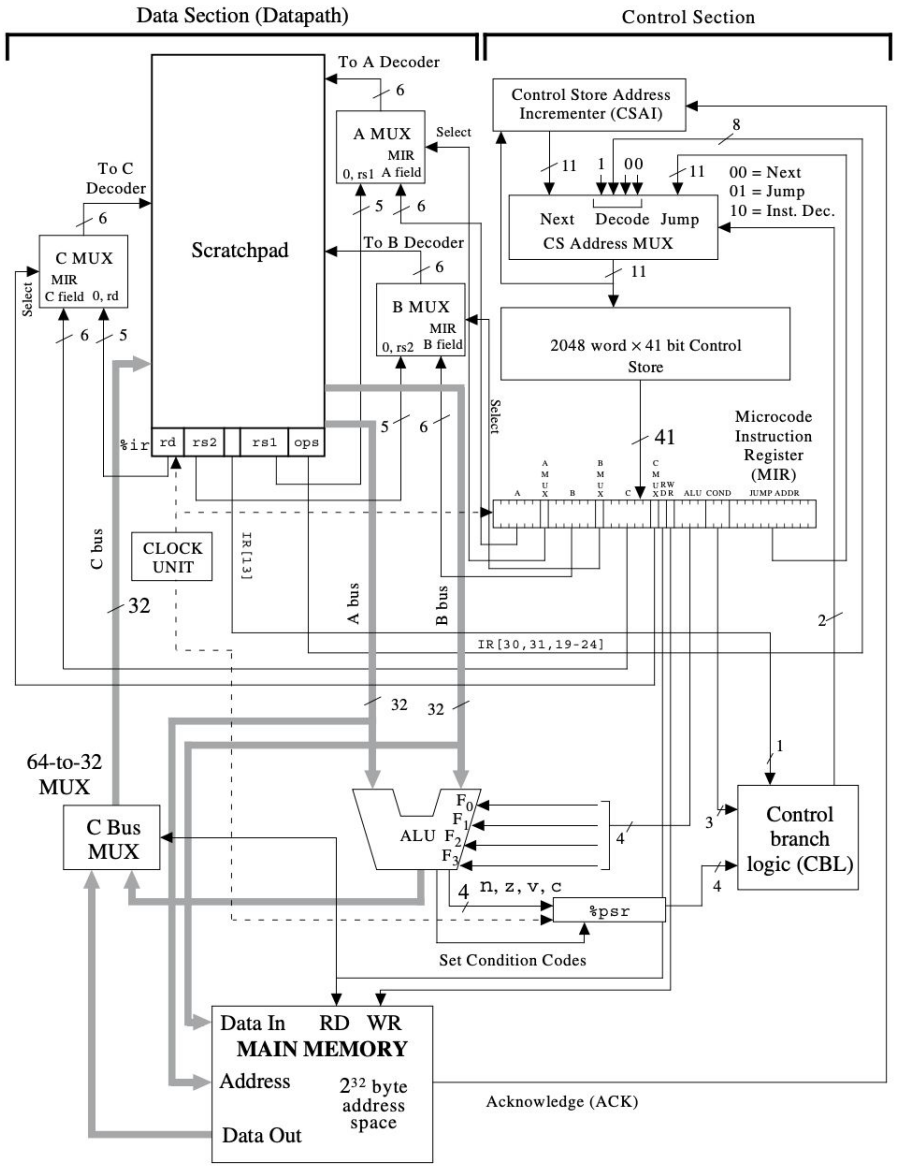


Ejecución

addcc %r10, 48, %r1

```
1602: R[temp0] ← SEXT13(R[ir]);
```

A	AMUX	B	BMUX	C	CMUXRW	ALU	COND	JUMP ADDR
000000	1	000000	1	000000	1 00	0011	110	11111111111

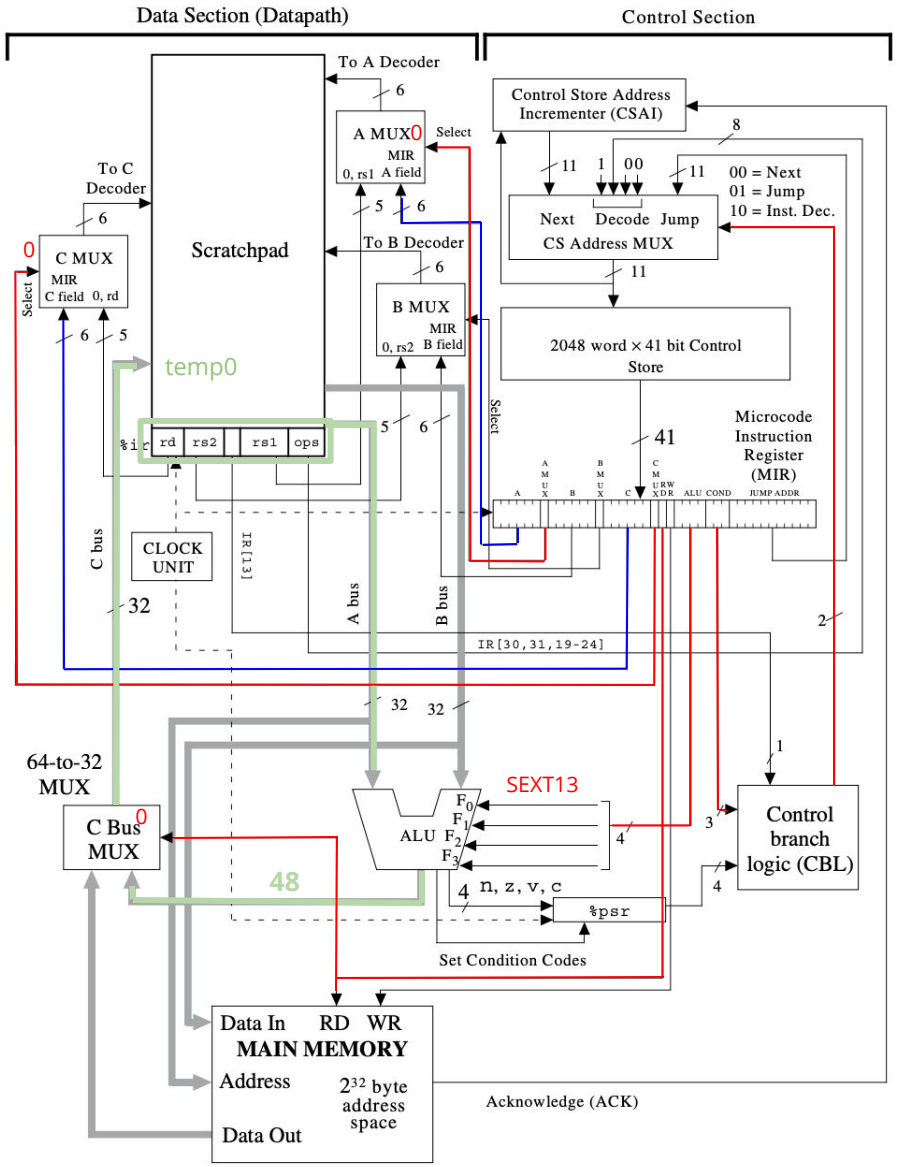


EJECUCIÓN

addcc %r10, 48, %r1

```
1602: R[temp0] ← SEXT13(R[ir]);
```

A	AMUX	B	BMUX	C	CMUXDR	ALU	COND	JUMP ADDR
100101	0	0000000	0	100001	000	1100	000	00000000000

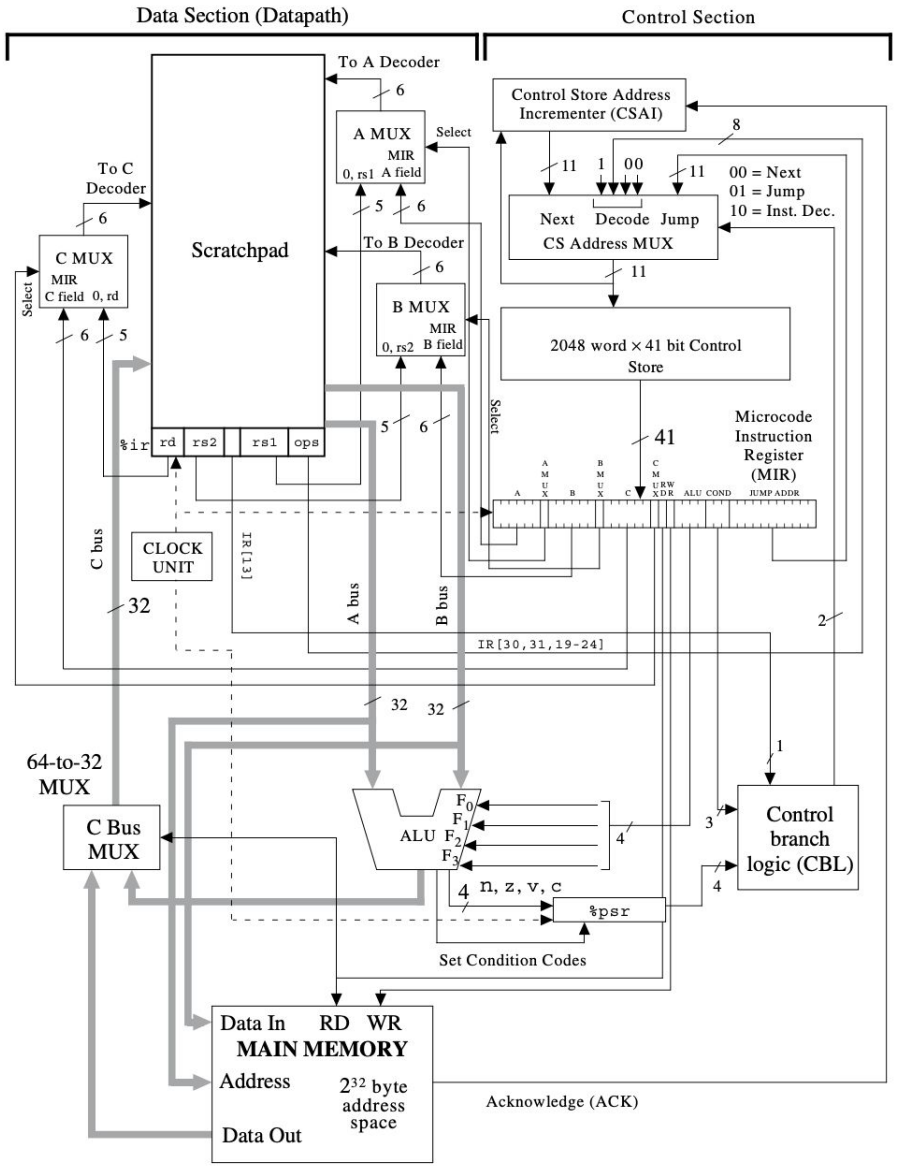


Ejecución

addcc %r10, 48, %r1

```
1603: R[rd] ← ADDCC(R[rs1],R[temp0]);
      GOTO 2047;
```

A	AMUX	B	BMUX	C	CMUXRW	ALU	COND	JUMP ADDR
000000	1	100001	0	000000	100	0011	110	11111111111

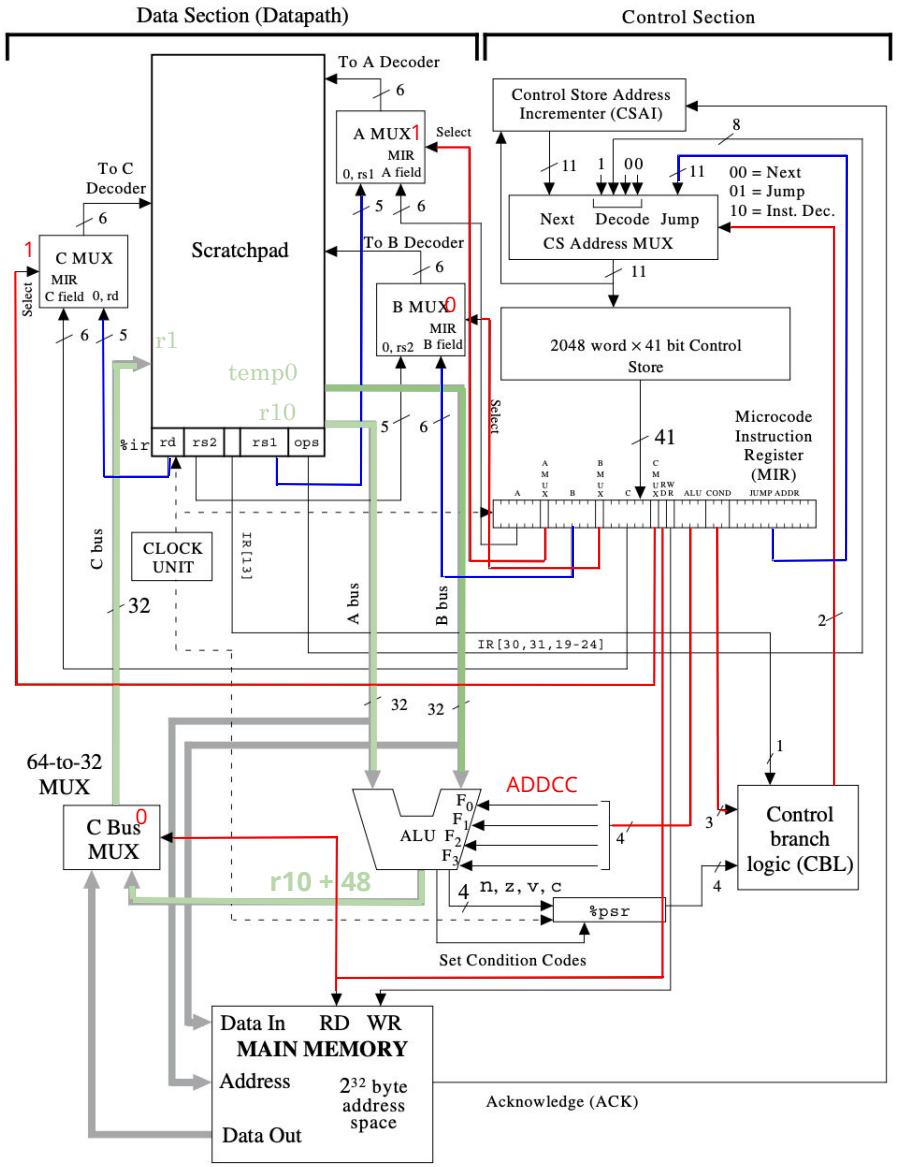


Ejecución

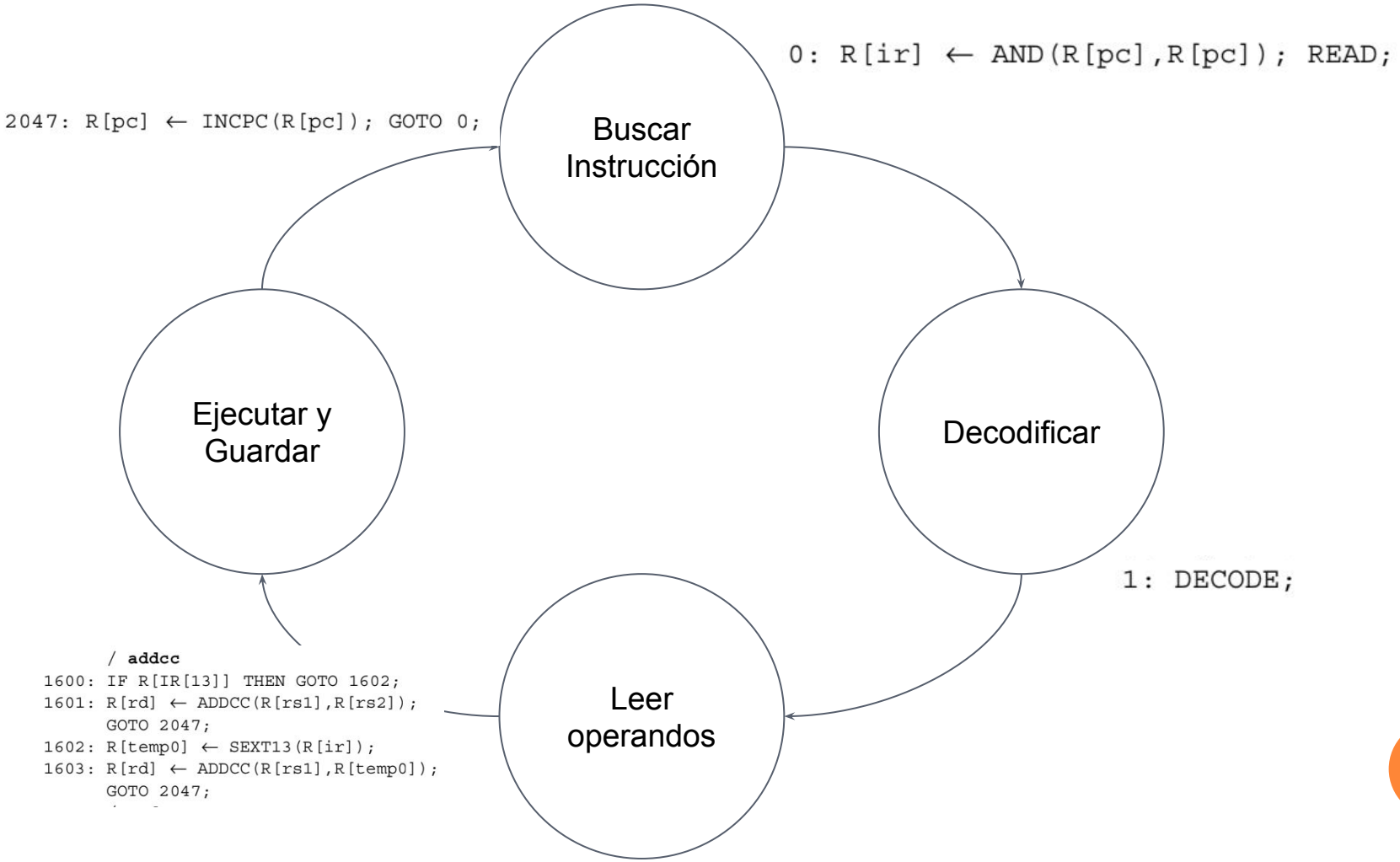
addcc %r10, 48, %r1

```
1603: R[rd] ← ADDCC(R[rs1],R[temp0]);
      GOTO 2047;
```

A	AMUX	B	BMUX	C	CMUXRW	ALU	COND	JUMP ADDR
000000	1	100001	0	000000	100	0011	110	11111111111



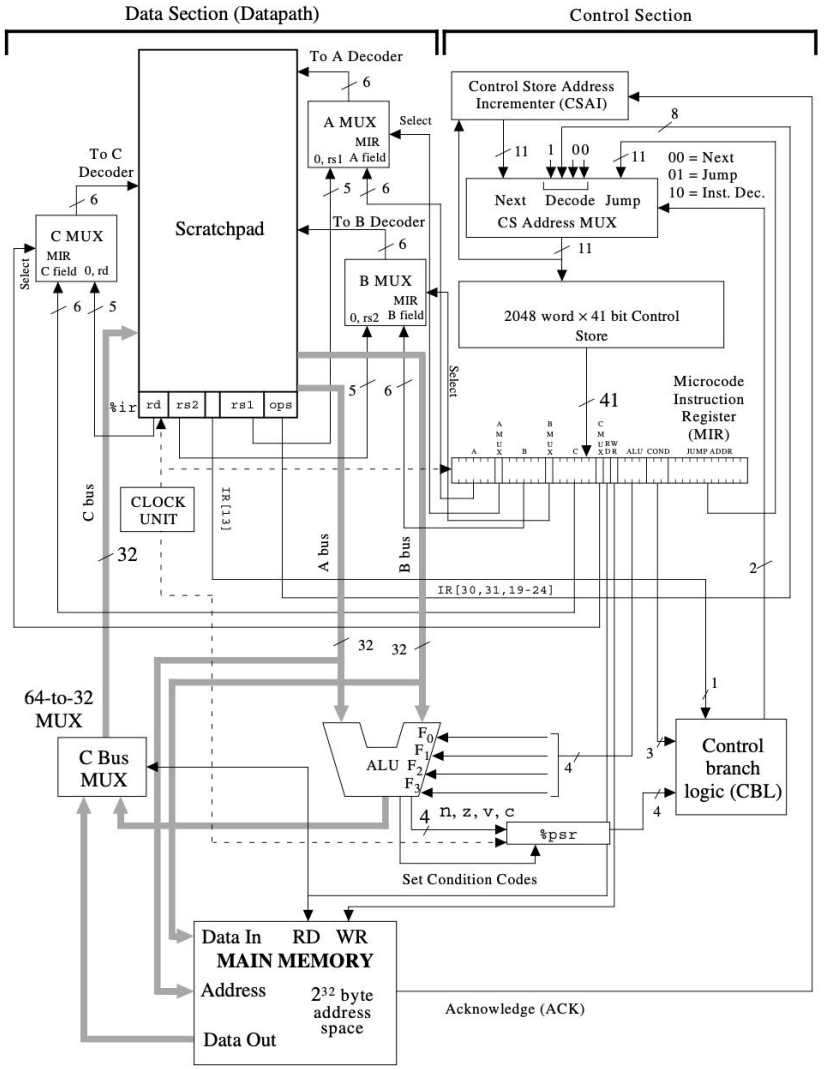
Ciclo de búsqueda

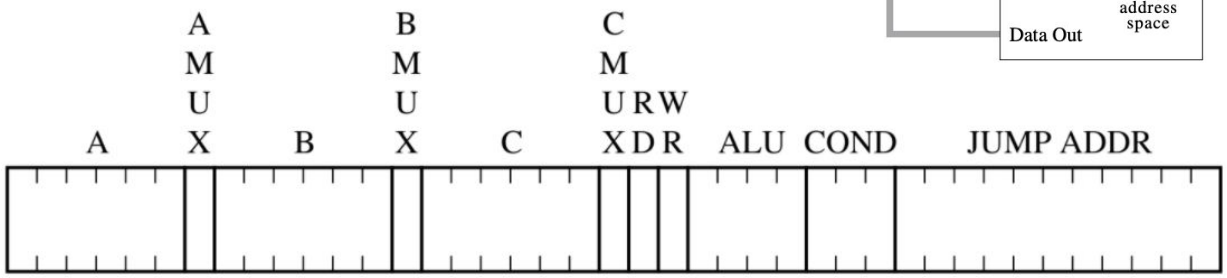


Buscar Instrucción

```
0: R[ir] ← AND(R[pc],R[pc]); READ;
1: DECODE;
```

```
2047: R[pc] ← INCPC(R[pc]); GOTO 0;
```





¿Preguntas?