Informe Etapa 1 - G11

Arquitectura de Computadores - 2021’2

Estructura de la Palabra

“10001100011000110001100011000110001100010”

- 7 menos significativos son para Opcode
- 16 más significativos son para literales
- 12 más significativos son la instrucción para el PC

Tabla

Instrucción	Operandos	Opcode	Condition	LoadPC	enableA	enableB	selA Zero = 00 A = 01 One = 11	selB Zero = 00 B = 01 Dout = 10 Lit = 11	selALU	Sadd0,1	Sdin0	Spc0	w	IncSp	DecSp
MOV	A,B	0000000		0	1	0	Zero	B	ADD	-	-	-	0	0	0
	B,A	0000001		0	0	1	A	Zero	ADD	-	-	-	0	0	0
	A,Lit	0000010		0	1	0	Zero	LIT	ADD	-	-	-	0	0	0
	B,Lit	0000011		0	0	1	Zero	LIT	ADD	-	-	-	0	0	0
	A,(Dir)	0000100		0	1	0	Zero	DOUT	ADD	LIT	-	-	0	0	0
	B,(Dir)	0000101		0	0	1	Zero	DOUT	ADD	LIT	-	-	0	0	0
	(Dir),A	0000110		0	0	0	A	ZERO	ADD	LIT	ALU	-	1	0	0
	(Dir),B	0000111		0	0	0	Zero	B	ADD	LIT	ALU	-	1	0	0
ADD	A,B	0001000		0	1	0	A	B	ADD	-			0		

	B,A	0001001		0	0	1	A	B	ADD	-			0		
	A,Lit	0001010		0	1	0	A	LIT	ADD	-			0		
	B,Lit	0001011		0	0	1	A	LIT	ADD				0		
	A,(Dir)	0001100		0	1	0	A	DOUT	ADD	LIT			0		
	B,(Dir)	0001101		0	0	1	A	DOUT	ADD				0		
	(Dir)	0001110		0	0	0	A	B	ADD	LIT			1		
SUB	A,B	0001111		0	1	0	A	B	SUB	-		-	0	0	0
	B,A	0010000		0	0	1	A	B	SUB	-		-	0	0	0
	A,Lit	0010001		0	1	0	A	LIT	SUB	-		-	0	0	0
	B,Lit	0010010		0	0	1	A	LIT	SUB	-		-	0	0	0
	A,(Dir)	0010011		0	1	0	A	DOUT	SUB	LIT		-	0	0	0
	B,(Dir)	0010100		0	0	1	A	DOUT	SUB			-	0	0	0
	(Dir)	0010101		0	0	0	A	B	SUB	LIT	ALU	-	1	0	0
AND	A,B	0010110		0	1	0	A	B	AND	-			0	0	0
	B,A	0010111		0	0	1	A	B	AND	-			0	0	0
	A,Lit	0011000		0	1	0	A	LIT	AND	-			0	0	0
	B,Lit	0011001		0	0	1	A	LIT	AND				0	0	0
	A,(Dir)	0011010		0	1	0	A	DOUT	AND	LIT			0	0	0
	B,(Dir)	0011011		0	0	1	A	DOUT	AND				0	0	0
	(Dir)	0011100		0	0	0	A	B	AND	LIT	ALU		1	0	0
OR	A,B	0011101		0	1	0	A	B	OR	-			0	0	0

	B,A	0011110		0	0	1	A	B	OR	-			0	0	0
	A,Lit	0011111		0	1	0	A	LIT	OR	-			0	0	0
	B,Lit	0100000		0	0	1	A	LIT	OR				0	0	0
	A,(Dir)	0100001		0	1	0	A	DOUT	OR	LIT			0	0	0
	B,(Dir)	0100010		0	0	1	A	DOUT	OR				0	0	0
	(Dir)	0100011		0	0	0	A	B	OR	LIT	ALU		1	0	0
XOR	A,B	0100100		0	1	0	A	B	XOR				0	0	0
	B,A	0100101		0	0	1	A	B	XOR				0	0	0
	A,Lit	0100110		0	1	0	A	LIT	XOR				0	0	0
	B,Lit	0100111		0	0	1	A	LIT	XOR				0	0	0
	A,(Dir)	0101000		0	1	0	A	DOUT	XOR	LIT			0	0	0
	B,(Dir)	0101001		0	0	1	A	DOUT	XOR				0	0	0
	(Dir)	0101010		0	0	0	A	B	XOR	LIT	ALU		1	0	0
NOT	A,A	0101011		0	1	0	A	-	NOT				0	0	0
	B,A	0101100		0	0	1	A	-	NOT				0	0	0
	(Dir),A	0101101		0	0	0	A	-	NOT	LIT	ALU		1	0	0
SHL	A,A	0101110		0	1	0	A	-	SHL				0		
	B,A	0101111		0	0	1	A	-	SHL				0		
	(Dir),A	0110000		0	0	0	A	-	SHL				1		
SHR	A,A	0110001		0	1	0	A	-	SHR				0	0	0
	B,A	0110010		0	0	1	A	-	SHR				0	0	0

	(Dir),A	0110011		0	0	0	A	-	SHR	LIT	ALU		1	0	0
INC	A	0110100		0	1	0	A	LIT (1)	ADD				0		
	B	0110101		0	0	1	ONE	B	ADD				0	0	0
	(Dir)	0110110		0	0	0	ONE	DOUT	ADD				1		
DEC	A	0110111		0	1	0	A	LIT (1)	SUB				0		
CMP	A,B	0111000		0	0	0	A	B	SUB				0	0	0
	A,Lit	0111001		0	0	0	A	LIT	SUB				0	0	0
	A,(Dir)	0111010		0	0	0	A	DOUT	SUB				0		
JMP	Ins	0111011		1	0	0						LIT	0	0	0
JEQ	Ins	0111100	Z=1	1	0	0						LIT	0	0	0
JNE	Ins	0111101	Z=0	1	0	0						LIT	0	0	0
NOP		0111110		0	1	0	A	Zero	ADD				0		

Rutina de Multiplicación → **Multiplicación Rusa** (a es b y b es a)

// Rutina de multiplicacion rusa (basada en la ayudantia 1)

// En breve, la idea es chequear si el primer operando es impar, si es impar, al resultado final se le suma el segundo operando, luego multiplicar el segundo operando por 2, dividir el primer operando por 2 y repetir

```
DATA:
mult_a          0
mult_b          0
mult_r          0

CODE:
MOV A, LIT              // Setea A
MOV B, LIT              // Setea B
MOV (mult_a), A         // Mueve el valor del registro A (primer operando) a la variable en memoria mult_a
MOV (mult_b), B         // “”
MOV B, 0                // B = 0
MOV (mult_r), B         // Mueve el valor del registro B a la variable en memoria mult_r, que corresponde al primer resultado parcial (X*0=0)

mult_loop:              // Label para poder hacer jumps y ejecutar la suma varias veces
MOV A, (mult_a)         // Mueve el valor de la variable en memoria mult_a al registro A
CMP A,0                 // Caso Base: Compara A con cero. Con todos los SHR, en algun momento A llega a 0
JEQ mult_end            // Si A es cero, salta a mult_end (termina el loop)
AND A,1                 // Si A es par, A=0. Si A es impar, A=1
CMP A,0                 // Compara A (ahora booleano si primer operando es impar) con cero
JEQ no_sum              // Si A es cero, salta a no_sum. Solamente pasa cuando el valor actual del primer operando es par
MOV A, (mult_r)         // Mueve el valor de la variable en memoria mult_r al registro A
MOV B, (mult_b)         // Mueve el valor de la variable en memoria mult_b al registro B
ADD (mult_r)            // Mueve a la variable en memoria mult_r la suma entre A y B

no_sum:                 // Label para el jump y, asi, evitar sumar cuando el primer operando es par
MOV A, (mult_a)         // Mueve el valor de la variable en memoria mult_a al registro A
SHR (mult_a)            // Mueve a la variable en memoria mult_a el valor de lo que hay en el registro A dividido en 2 (‘0’&mult_a[:-1])
MOV A, (mult_b)         // Mueve el valor de la memoria mult_b al registro A
SHL (mult_b)            // Mueve a la variable en memoria mult_b el valor de lo que hay en el registro A multiplicado por 2
JMP mult_loop           // Volvemos al label mult_loop para seguir en el loop
```

```
mult_end:                   // Label para el jump y terminar la multiplicación
MOV A, (mult_r)             // Mueve el valor de la variable en memoria mult_r al registro A
```