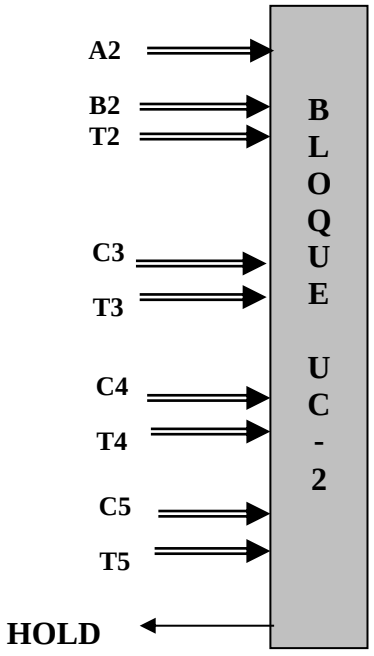
Procesador EV-20

El procesador **EV 20** es un procesador programable de alta performance con estructura *pipeline* de 5 etapas. Cuenta con memoria de programa separada de la memoria de datos. Tiene 28 registros de propósito general (R0 a R27), dos Puertos de Entrada PI 0/1 ("mapeados" como registros R28/R29), dos Puertos de Salida PO 0/1 ("mapeados" como registros R30/R31), y un registro especial (W: Working Register) sobre el que convergen la mayoría de las operaciones lógico-aritméticas, y a través del cual se realizan los movimientos de datos a memoria.

El procesador **EV20** tiene un Conjunto Reducido de Instrucciones (RISC) implementado mediante una microarquitectura basada en microcódigo. Las instrucciones y el hardware han sido diseñados de modo que por cada instrucción exista una única microinstrucción, para así aumentar la eficacia. Las microinstrucciones se almacenan en una Micro-Operation ROM que es indexada con el código de operación (opcode) de la instrucción actualmente en curso. Esta Micro-Operation ROM provee la microinstrucción utilizada por los registros de microinstrucción (MIRs) del pipeline. El banco de registros cuenta con dos registros auxiliares (Aux.0 y Aux.1) no accesibles al programador.

La **Tabla A** muestra el conjunto de instrucciones junto con el correspondiente código binario, y la **Tabla B** los campos de la microinstrucción, donde figuran los bits **Ti** que definen el **Tipo** de instrucción, bits que son usados por la unidad de control para administrar la dependencia de registros. Estos bits $\mathbf{T_6..T_0}$ identifican lectura y escritura de registro W, lectura y escritura de un registro genérico Ri, Lectura y Escritura de Carry, y Salto.

Las **Figuras A y B** muestran el diagrama en bloques de la arquitectura de este procesador, en el que se han representado dos bloques de la Unidad de Control (**UC-1** y **UC-2**) utilizados para administrar todo tipo de dependencias, en particular la **Dependencia de Registros**. La Unidad de Control está diseñada para no bloquear innecesariamente el progreso del pipeline y genera una acción de **HW_Not_Operate** consistente



El EV20 es una evolución de los modelos EV*xx* anteriores, e incorpora registros de 16 bits y una salida VGA que maneja gráficos en baja o media resolución (320 x 200, 8 colores como mínimo). El set de instrucciones debe ser adaptado para manejar constantes de 16 bits. Se puede ampliar la memoria de programa. EV20 implementa el manejo de subrutinas con un stack de direcciones de retorno que tiene al menos cuatro niveles, implementado mediante hardware específico.

Cada equipo de trabajo deberá proponer a la cátedra un diseño de arquitectura basado en EV20, con sus propias características y funcionalidades adicionales. Una vez aprobada por la cátedra esta especificación, deberán implementarlo en FPGA y demostrar su funcionamiento mediante uno o más programas que puedan ser modificados o parametrizados "en vivo" durante la demostración en el laboratorio. Se intenta limitar el empleo de herramientas de tipo VHDL o equivalentes que podrán usarse solamente en casos puntuales y previamente acordados con la cátedra. El enfoque general apunta a la implementación del procesador con bloques de funciones lógicas (registros, ALU, etc.).

Código Binario de Instrucción EV16	Nemónico	Instrucción	Significado
1 0 0 x x x x x x x x x x x x	JMP X	Unconditional Jump	PC = X
1 0 1 x x x x x x x x x x x x	JZE X	Jump if Working Register is Zero	IF W=0 THEN PC=X
1 1 0 x x x x x x x x x x x x	JNE X	Jump if Working Register is Negative	IF W_{15}=0 THEN PC=X
1 1 1 x x x x x x x x x x x x	JCY X	Jump if Carry	IF CY THEN PC=X
0 1 0 0 y y y y y y y y y y y	MOM Y,W	Move Working Register to Memory	M(Y) = W
0 1 0 1 y y y y y y y y y y y	MOM W,Y	Move Memory to Working Register	W = M(Y)
0 1 1 0 i i i i i j j j j j	ADW Ri,Rj	Add with Carry Reg. J with Working Reg. to Reg. I	Ri= W + Rj + CY
0 1 1 1 s s s s s s s s s s	BSR S	Unconditional Branch (Relative) to Subroutine S	Save PC; PC= PC + S
0 0 1 0 i i i i i j j j j j	MOV Ri,Rj	Move Register J to Register I {Ri, Rj: 0 a 27}	Ri = Rj
0 0 1 0 1 1 1 1 i j j j j j	MOV POi,Rj	Move Register J to Output Port i	POi = Rj
0 0 1 0 i i i i i 1 1 1 0 j	MOV Ri,PIj	Move Input Port j to Register I	Ri = PIj
0 0 1 0 1 1 1 1 i 1 1 1 0 j	MOV POi,PIj	Move Input Port J to Output Port I	POi = PIj
0 0 1 1 i i i i i 0 0 0 0 0	MOV Ri,W	Move Working Register to Register I	Ri = W
0 0 1 1 1 1 1 1 i 0 0 0 0 0	MOV POi,W	Move Working Register to Output Port I	POi = W
0 0 0 1 0 0 k k k k k k k k	MOK W,#K	Move Constant to Working Register	W = K
0 0 0 1 0 1 k k k k k k k k	ANK W,#K	AND Constant with Working Register	W= W & K
0 0 0 1 1 0 k k k k k k k k	ORK W,#K	OR Constant with Working Register	W= W OR K
0 0 0 1 1 1 k k k k k k k k	ADK W,#K	ADD with Carry Constant with Working Register	W= W + K + CY
0 0 0 0 1 0 0 0 0 j j j j j	MOV W,Rj	Move Register J to Working Register	W= Rj
0 0 0 0 1 0 0 0 0 1 1 1 0 j	MOV W,PIj	Move Input Port J to Working Register	W= PIj
0 0 0 0 1 0 1 0 0 j j j j j	ANR W,Rj	AND Register J with Working Register	W= W & Rj
0 0 0 0 1 1 0 0 0 j j j j j	ORR W,Rj	OR Register J with Working Register	W= W OR Rj
0 0 0 0 1 1 1 0 0 j j j j j	ADR W,Rj	ADD with Carry Register J with Working Register	W= W + Rj + CY
0 0 0 0 0 0 0 0 0 0 0 0 0 0	CPL W	Complement Working Register	W= /W
0 0 0 0 0 0 1 0 0 0 0 0 0 0	CLR CY	Clear Carry	CY= 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0	SET CY	Set Carry	CY= 1
0 0 0 0 0 1 1 0 0 0 0 0 0 0	RET	Return From Subroutine	PC= Latest Stored PC {+ 1}

Nemónico	Significado	Función ALU	ALUC (4)	SH (2)	KMx (1)	MR (1)	MW (1)	B Bus (6)	C Bus (6)	TYPE (6) $T_6..T_0$	A Bus (5)
JMP X	PC = X	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1000000**	**+**
JZE X	IF W=0 THEN PC=X	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1000001**	**+**
JNE X	IF W_{15}=0 THEN PC=X	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1000001**	**+**
JCY X	IF CY THEN PC=X	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1010000**	**+**
MOM Y,W	M(Y) = W	**-**	**0**	**0**	**0**	**0**	**1**	**-**	**35**	**0000001**	**+**
MOM W,Y	W = M(Y)	**-**	**0**	**0**	**0**	**1**	**0**	**-**	**35**	**0000010**	**+**
ADW Ri,Rj	Ri= W + Rj + CY	**A+B+Cy**	**5**	**0**	**0**	**0**	**0**	**34**	**Ri**	**0111101**	**Rj**
BSR S	Save PC; PC= PC + S	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1000000**	**+**
MOV Ri,Rj	Ri = Rj	**A**	**0**	**0**	**0**	**0**	**0**	**-**	**Ri**	**0001100**	**Rj**
MOV POi,Rj	POi = Rj	**A**	**0**	**0**	**0**	**0**	**0**	**-**	**POi**	**0001100**	**Rj**
MOV Ri,POj	Ri = POj	**A**	**0**	**0**	**0**	**0**	**0**	**-**	**Ri**	**0001100**	**POj**
MOV Ri,W	Ri = W	**B**	**1**	**0**	**0**	**0**	**0**	**34**	**Ri**	**0001001**	**+**
MOV POi,W	POi = W	**B**	**1**	**0**	**0**	**0**	**0**	**34**	**POi**	**0001001**	**+**
MOK W,#K	W = K	**A**	**0**	**0**	**1**	**0**	**0**	**-**	**34**	**0000010**	**+**
ANK W,#K	W= W & K	**A&B**	**7**	**0**	**1**	**0**	**0**	**34**	**34**	**0000011**	**+**
ORK W,#K	W= W OR K	**Corb**	**6**	**0**	**1**	**0**	**0**	**34**	**34**	**0000011**	**+**
ADK W,#K	W= W + K + CY	**A+B+Cy**	**5**	**0**	**1**	**0**	**0**	**34**	**34**	**0110011**	**+**
MOV W,Rj	W= Rj	**A**	**0**	**0**	**0**	**0**	**0**	**-**	**34**	**0000110**	**Rj**
MOV W,PIj	W= PIj	**A**	**0**	**0**	**0**	**0**	**0**	**-**	**34**	**0000110**	**PIj**
ANR W,Rj	W= W & Rj	**A&B**	**7**	**0**	**0**	**0**	**0**	**34**	**34**	**0000111**	**Rj**
ORR W,Rj	W= W OR Rj	**Corb**	**6**	**0**	**0**	**0**	**0**	**34**	**34**	**0000111**	**Rj**
ADR W,Rj	W= W + Rj + CY	**A+B+Cy**	**5**	**0**	**0**	**0**	**0**	**34**	**34**	**0110111**	**Rj**
CPL W	W= /W	**/B**	**3**	**0**	**0**	**0**	**0**	**34**	**34**	**0000011**	**+**
CLR CY	CY= 0	**-**	**11**	**0**	**0**	**0**	**0**	**-**	**35**	**0100000**	**+**
SET CY	CY= 1	**-**	**12**	**0**	**0**	**0**	**0**	**-**	**35**	**0100000**	**+**
RET	PC= Last Stored PC+1	**-**	**0**	**0**	**0**	**0**	**0**	**-**	**35**	**1000000**	**+**

Tabla A: Campos de la Microinstrucción

NOTAS

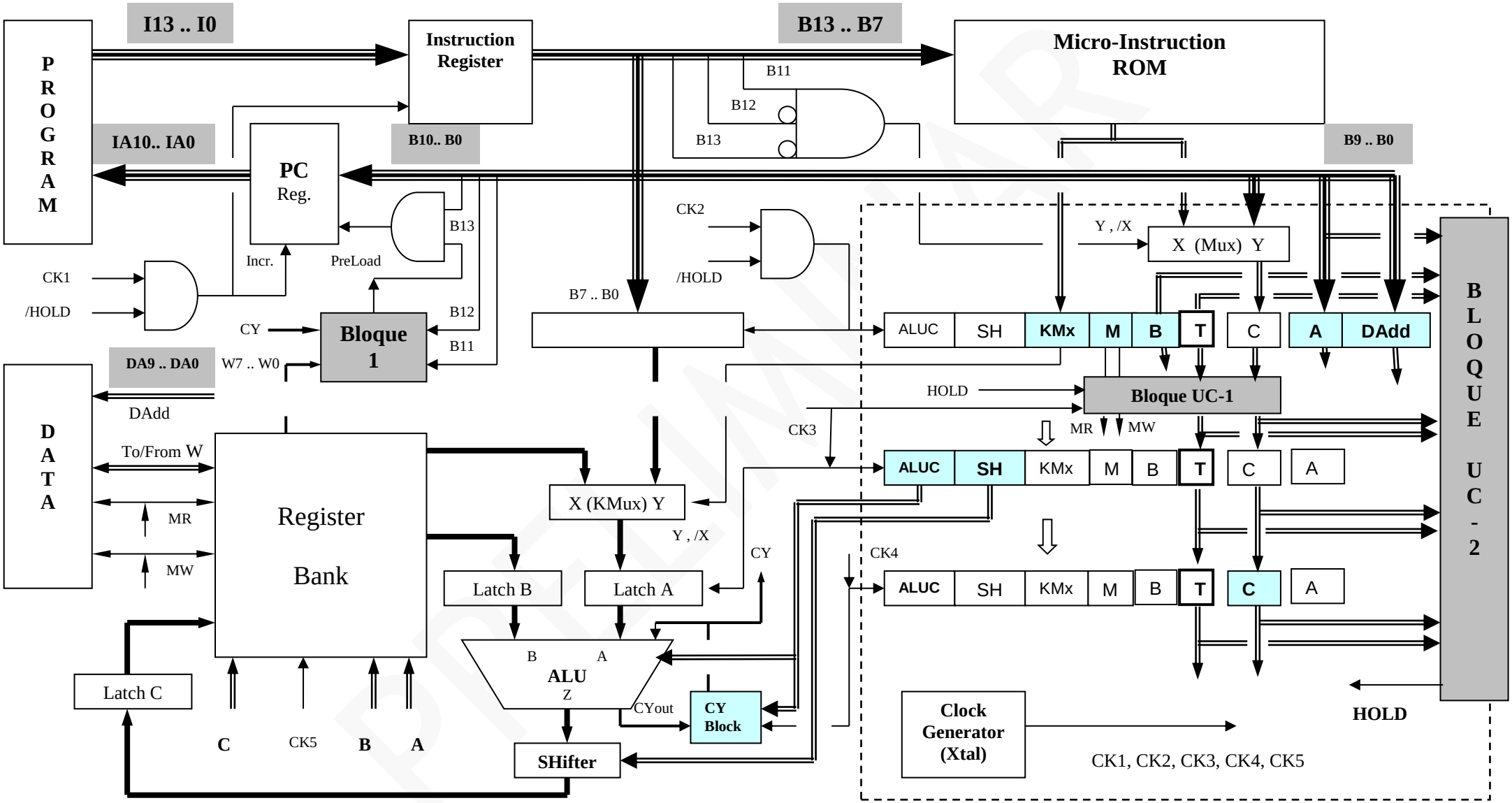
Los valores en columnas ALUC indican el código de la operación ALU que involucrará o no modificación del estado del Carry.

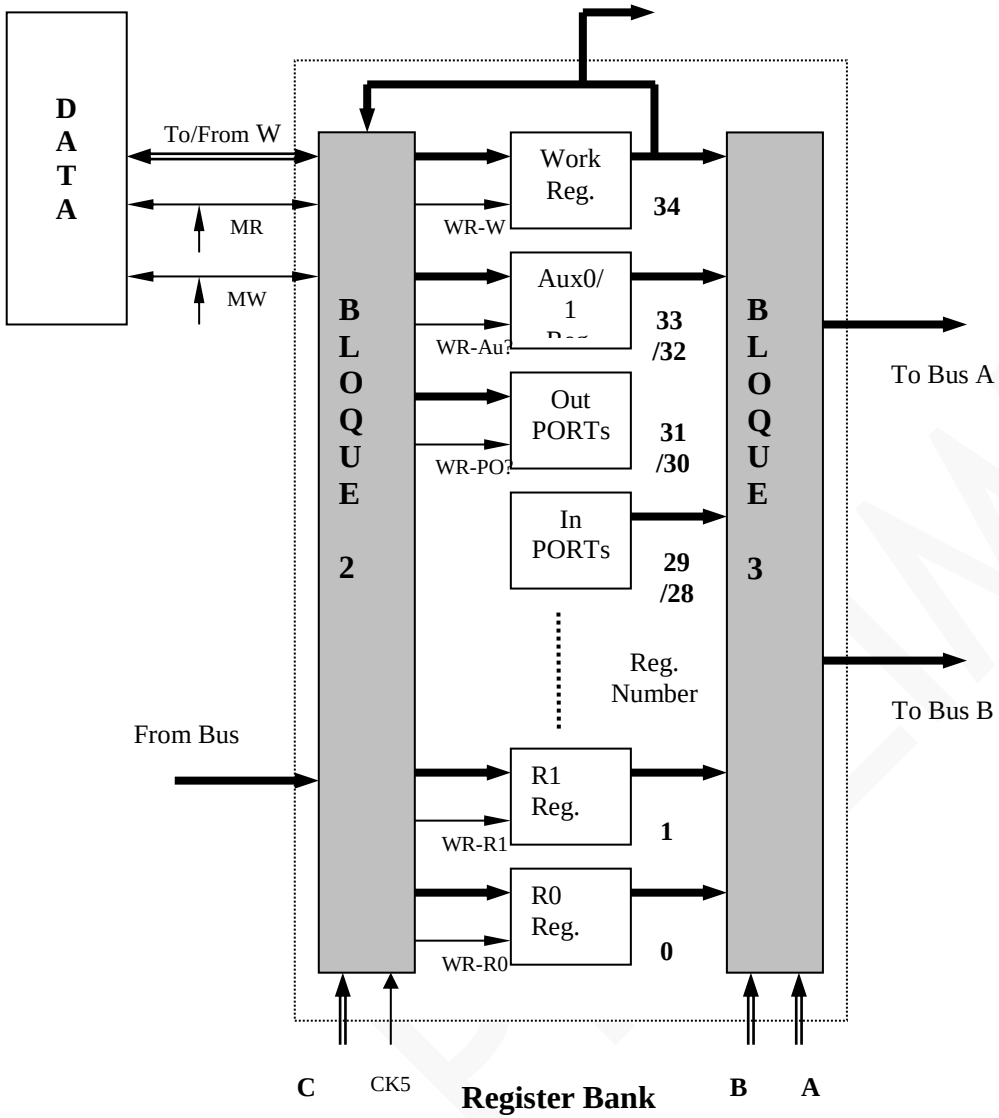
El dato decimal 35 indica acceso a registro inexistente (garantiza que no se escriba ni lea ningún registro del banco).

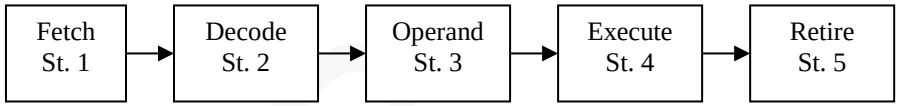
El símbolo (+) indica que los valores son irrelevantes y corresponden a los 5 bits menos significativos de la instrucción.

El símbolo (-) indica que el valor es irrelevante. Puede ser tomado como "0" si así se desea.

Ri, Rj toman valores de 0 a 27, PI0= 28, PI1= 29, PO0= 30, PO1= 31.


PROGRAM
I13 .. I0
Instruction Register
B13 .. B7
Micro-Instruction ROM
B11
B12
B13
IA10.. IA0
PC Reg.
B10.. B0
B9 .. B0
Incr.
PreLoad
B13
CK1
/HOLD
CY
Bloque 1
B12
B11
CK2
/HOLD
B7 .. B0
Y , /X
X (Mux) Y
ALUC
SH
KMx
M
B
T
C
A
DAdd
DA9 .. DA0
W7 .. W0
DAdd
HOLD
Bloque UC-1
MR
MW
CK3
DATA
To/From W
MR
MW
Register Bank
X (KMux) Y
Y , /X
CY
CK4
Latch B
Latch A
B
A
ALU
Z
CYout
CY Block
Latch C
C
CK5
B
A
SHifter
Clock Generator (Xtal)
CK1, CK2, CK3, CK4, CK5
HOLD
BLOQUE UC-2





ALU-CY Operations:
0: Z= A
1: Z= B
2: Z= /A
3: Z= /B
4: Z= A+B **
5: Z= A+B+CY **
6: Z= A OR B
7: Z= A & B
8: Z= 0
9: Z= 1
10: Z= 0xFFFF
11: CY= 0 **
12: CY= 1 **

SHifter Operations:
0: No Shift
1: Shift Right 1 Bit
2: Shift Left 1 Bit

**: CY is updated

NOTES:
Ix: Instruction
IAx: Instruction Address
All latches´clock inputs are active on clock rising edge