Arquitectura de procesadores

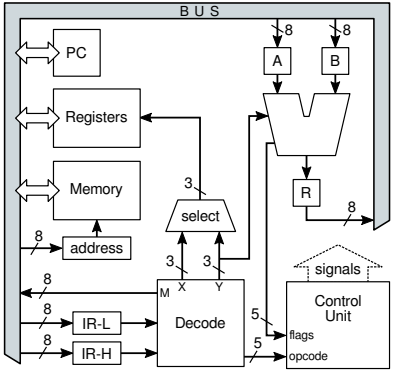
David Alejandro González Márquez

Clase disponible en: `https://github.com/fokerman/computingSystemsCourse`

Arquitectura OrgaSmall

Para ilustrar el funcionamiento de un sistema de computo, vamos a presentar la arquitectura *OrgaSmall*.

Esta arquitectura fue diseñada con fines didácticos y su objetivo es poner de manifiesto todos los conceptos básicos de la organización de un procesador con una arquitectura simple.



- 8 registros de propósito general, R0 a R7.
- 1 registro de propósito específico PC.
- Unidad direccionable de 8 bits (1 byte) e instrucciones de 16 bits (2 bytes).
- Memoria de 256 bytes.
- Bus de 8 bits.
- Diseño microprogramado.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Instrucción	Acción	Codificación
Reservada	Codifica el *Fetch* de instrucciones	00000----------
ADD Rx, Ry	Rx ← Rx + Ry	00001XXXYYY-----
ADC Rx, Ry	Rx ← Rx + Ry + Flag_C	00010XXXYYY-----
SUB Rx, Ry	Rx ← Rx - Ry	00011XXXYYY-----
AND Rx, Ry	Rx ← Rx and Ry	00100XXXYYY-----
OR Rx, Ry	Rx ← Rx or Ry	00101XXXYYY-----
XOR Rx, Ry	Rx ← Rx xor Ry	00110XXXYYY-----
CMP Rx, Ry	Modifica flags de Rx - Ry	00111XXXYYY-----
MOV Rx, Ry	Rx ← Ry	01000XXXYYY-----
PUSH |Rx|, Ry	Mem[Rx] ← Ry ; Rx ← Rx-1	01001XXXYYY-----
POP |Rx|, Ry	Rx ← Rx+1 ; Ry ← Mem[Rx]	01010XXXYYY-----
CALL |Rx|, Ry	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	01011XXXYYY-----
CALL |Rx|, M	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	01100XXXMMMMMMMM
RET |Rx|	Rx ← Rx+1 ; PC ← Mem[Rx]	01101XXX--------
Libre		01110----------
Libre		01111----------
STR [M], Rx	Mem[M] ← Rx	10000XXXMMMMMMMM
LOAD Rx, [M]	Rx ← Mem[M]	10001XXXMMMMMMMM
STR [Rx], Ry	Mem[Rx] ← Ry	10010XXXYYY-----
LOAD Rx, [Ry]	Rx ← Mem[Ry]	10011XXXYYY-----
JMP M	PC ← M	10100---MMMMMMMM
JC M	Si flag_C=1 entonces PC ← M	10101---MMMMMMMM
JZ M	Si flag_Z=1 entonces PC ← M	10110---MMMMMMMM
JN M	Si flag_N=1 entonces PC ← M	10111---MMMMMMMM
JO M	Si flag_O=1 entonces PC ← M	11000---MMMMMMMM
SHRA Rx, t	Rx ← Rx >>> t	11001XXXYYY-----
SHR Rx, t	Rx ← Rx >> t	11010XXXYYY-----
SHL Rx, t	Rx ← Rx << t	11011XXXYYY-----
READF Rx	Rx ← Flags	11100XXX--------
LOADF Rx	Flags ← Rx	11101XXX--------
SET Rx, M	Rx ← M	11110XXXMMMMMMMM
Libre		11111----------

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Ejemplos

Instrucción	Codificación
`ADD R3, R6`	
`STR [0x21], R5`	
`SET R1, 0xFF`	
`SHL R7, 3`	

Instrucción	Acción	Codificación
`Reservada`	Codifica el *Fetch* de instrucciones	`00000----------`
`ADD Rx, Ry`	Rx ← Rx + Ry	`00001XXXYYY-----`
`ADC Rx, Ry`	Rx ← Rx + Ry + Flag_C	`00010XXXYYY-----`
`SUB Rx, Ry`	Rx ← Rx - Ry	`00011XXXYYY-----`
`AND Rx, Ry`	Rx ← Rx and Ry	`00100XXXYYY-----`
`OR Rx, Ry`	Rx ← Rx or Ry	`00101XXXYYY-----`
`XOR Rx, Ry`	Rx ← Rx xor Ry	`00110XXXYYY-----`
`CMP Rx, Ry`	Modifica flags de Rx - Ry	`00111XXXYYY-----`
`MOV Rx, Ry`	Rx ← Ry	`01000XXXYYY-----`
`PUSH |Rx|, Ry`	Mem[Rx] ← Ry ; Rx ← Rx-1	`01001XXXYYY-----`
`POP |Rx|, Ry`	Rx ← Rx+1 ; Ry ← Mem[Rx]	`01010XXXYYY-----`
`CALL |Rx|, Ry`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	`01011XXXYYY-----`
`CALL |Rx|, M`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	`01100XXXMMMMMMMM`
`RET |Rx|`	Rx ← Rx+1 ; PC ← Mem[Rx]	`01101XXX--------`
`Libre`		`01110----------`
`Libre`		`01111----------`
`STR [M], Rx`	Mem[M] ← Rx	`10000XXXMMMMMMMM`
`LOAD Rx, [M]`	Rx ← Mem[M]	`10001XXXMMMMMMMM`
`STR [Rx], Ry`	Mem[Rx] ← Ry	`10010XXXYYY-----`
`LOAD Rx, [Ry]`	Rx ← Mem[Ry]	`10011XXXYYY-----`
`JMP M`	PC ← M	`10100---MMMMMMMM`
`JC M`	Si flag_C=1 entonces PC ← M	`10101---MMMMMMMM`
`JZ M`	Si flag_Z=1 entonces PC ← M	`10110---MMMMMMMM`
`JN M`	Si flag_N=1 entonces PC ← M	`10111---MMMMMMMM`
`JO M`	Si flag_O=1 entonces PC ← M	`11000---MMMMMMMM`
`SHRA Rx, t`	Rx ← Rx >>> t	`11001XXXYYY-----`
`SHR Rx, t`	Rx ← Rx >> t	`11010XXXYYY-----`
`SHL Rx, t`	Rx ← Rx << t	`11011XXXYYY-----`
`READF Rx`	Rx ← Flags	`11100XXX--------`
`LOADF Rx`	Flags ← Rx	`11101XXX--------`
`SET Rx, M`	Rx ← M	`11110XXXMMMMMMMM`
`Libre`		`11111----------`

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Ejemplos

Instrucción	Codificación
`ADD R3, R6`	`0000101111000000`
`STR [0x21], R5`	
`SET R1, 0xFF`	
`SHL R7, 3`	

Instrucción	Acción	Codificación
`Reservada`	Codifica el *Fetch* de instrucciones	`00000----------`
`ADD Rx, Ry`	Rx ← Rx + Ry	`00001XXXYYY-----`
`ADC Rx, Ry`	Rx ← Rx + Ry + Flag_C	`00010XXXYYY-----`
`SUB Rx, Ry`	Rx ← Rx - Ry	`00011XXXYYY-----`
`AND Rx, Ry`	Rx ← Rx and Ry	`00100XXXYYY-----`
`OR Rx, Ry`	Rx ← Rx or Ry	`00101XXXYYY-----`
`XOR Rx, Ry`	Rx ← Rx xor Ry	`00110XXXYYY-----`
`CMP Rx, Ry`	Modifica flags de Rx - Ry	`00111XXXYYY-----`
`MOV Rx, Ry`	Rx ← Ry	`01000XXXYYY-----`
`PUSH |Rx|, Ry`	Mem[Rx] ← Ry ; Rx ← Rx-1	`01001XXXYYY-----`
`POP |Rx|, Ry`	Rx ← Rx+1 ; Ry ← Mem[Rx]	`01010XXXYYY-----`
`CALL |Rx|, Ry`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	`01011XXXYYY-----`
`CALL |Rx|, M`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	`01100XXXMMMMMMMM`
`RET |Rx|`	Rx ← Rx+1 ; PC ← Mem[Rx]	`01101XXX--------`
`Libre`		`01110----------`
`Libre`		`01111----------`
`STR [M], Rx`	Mem[M] ← Rx	`10000XXXMMMMMMMM`
`LOAD Rx, [M]`	Rx ← Mem[M]	`10001XXXMMMMMMMM`
`STR [Rx], Ry`	Mem[Rx] ← Ry	`10010XXXYYY-----`
`LOAD Rx, [Ry]`	Rx ← Mem[Ry]	`10011XXXYYY-----`
`JMP M`	PC ← M	`10100---MMMMMMMM`
`JC M`	Si flag_C=1 entonces PC ← M	`10101---MMMMMMMM`
`JZ M`	Si flag_Z=1 entonces PC ← M	`10110---MMMMMMMM`
`JN M`	Si flag_N=1 entonces PC ← M	`10111---MMMMMMMM`
`JO M`	Si flag_O=1 entonces PC ← M	`11000---MMMMMMMM`
`SHRA Rx, t`	Rx ← Rx >>> t	`11001XXXYYY-----`
`SHR Rx, t`	Rx ← Rx >> t	`11010XXXYYY-----`
`SHL Rx, t`	Rx ← Rx << t	`11011XXXYYY-----`
`READF Rx`	Rx ← Flags	`11100XXX--------`
`LOADF Rx`	Flags ← Rx	`11101XXX--------`
`SET Rx, M`	Rx ← M	`11110XXXMMMMMMMM`
`Libre`		`11111----------`

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Ejemplos

Instrucción	Codificación
`ADD R3, R6`	`0000101111000000`
`STR [0x21], R5`	`1000010100100001`
`SET R1, 0xFF`	
`SHL R7, 3`	

Instrucción	Acción	Codificación
`Reservada`	Codifica el *Fetch* de instrucciones	`00000----------`
`ADD Rx, Ry`	Rx ← Rx + Ry	`00001XXXYYY-----`
`ADC Rx, Ry`	Rx ← Rx + Ry + Flag_C	`00010XXXYYY-----`
`SUB Rx, Ry`	Rx ← Rx - Ry	`00011XXXYYY-----`
`AND Rx, Ry`	Rx ← Rx and Ry	`00100XXXYYY-----`
`OR Rx, Ry`	Rx ← Rx or Ry	`00101XXXYYY-----`
`XOR Rx, Ry`	Rx ← Rx xor Ry	`00110XXXYYY-----`
`CMP Rx, Ry`	Modifica flags de Rx - Ry	`00111XXXYYY-----`
`MOV Rx, Ry`	Rx ← Ry	`01000XXXYYY-----`
`PUSH |Rx|, Ry`	Mem[Rx] ← Ry ; Rx ← Rx-1	`01001XXXYYY-----`
`POP |Rx|, Ry`	Rx ← Rx+1 ; Ry ← Mem[Rx]	`01010XXXYYY-----`
`CALL |Rx|, Ry`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	`01011XXXYYY-----`
`CALL |Rx|, M`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	`01100XXXMMMMMMMM`
`RET |Rx|`	Rx ← Rx+1 ; PC ← Mem[Rx]	`01101XXX--------`
`Libre`		`01110----------`
`Libre`		`01111----------`
`STR [M], Rx`	Mem[M] ← Rx	`10000XXXMMMMMMMM`
`LOAD Rx, [M]`	Rx ← Mem[M]	`10001XXXMMMMMMMM`
`STR [Rx], Ry`	Mem[Rx] ← Ry	`10010XXXYYY-----`
`LOAD Rx, [Ry]`	Rx ← Mem[Ry]	`10011XXXYYY-----`
`JMP M`	PC ← M	`10100---MMMMMMMM`
`JC M`	Si flag_C=1 entonces PC ← M	`10101---MMMMMMMM`
`JZ M`	Si flag_Z=1 entonces PC ← M	`10110---MMMMMMMM`
`JN M`	Si flag_N=1 entonces PC ← M	`10111---MMMMMMMM`
`JO M`	Si flag_O=1 entonces PC ← M	`11000---MMMMMMMM`
`SHRA Rx, t`	Rx ← Rx >>> t	`11001XXXYYY-----`
`SHR Rx, t`	Rx ← Rx >> t	`11010XXXYYY-----`
`SHL Rx, t`	Rx ← Rx << t	`11011XXXYYY-----`
`READF Rx`	Rx ← Flags	`11100XXX--------`
`LOADF Rx`	Flags ← Rx	`11101XXX--------`
`SET Rx, M`	Rx ← M	`11110XXXMMMMMMMM`
`Libre`		`11111----------`

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Ejemplos

Instrucción	Codificación
`ADD R3, R6`	`0000101111000000`
`STR [0x21], R5`	`1000010100100001`
`SET R1, 0xFF`	`1111000111111111`
`SHL R7, 3`	

Instrucción	Acción	Codificación
`Reservada`	Codifica el *Fetch* de instrucciones	`00000----------`
`ADD Rx, Ry`	Rx ← Rx + Ry	`00001XXXYYY-----`
`ADC Rx, Ry`	Rx ← Rx + Ry + Flag_C	`00010XXXYYY-----`
`SUB Rx, Ry`	Rx ← Rx - Ry	`00011XXXYYY-----`
`AND Rx, Ry`	Rx ← Rx and Ry	`00100XXXYYY-----`
`OR Rx, Ry`	Rx ← Rx or Ry	`00101XXXYYY-----`
`XOR Rx, Ry`	Rx ← Rx xor Ry	`00110XXXYYY-----`
`CMP Rx, Ry`	Modifica flags de Rx - Ry	`00111XXXYYY-----`
`MOV Rx, Ry`	Rx ← Ry	`01000XXXYYY-----`
`PUSH |Rx|, Ry`	Mem[Rx] ← Ry ; Rx ← Rx-1	`01001XXXYYY-----`
`POP |Rx|, Ry`	Rx ← Rx+1 ; Ry ← Mem[Rx]	`01010XXXYYY-----`
`CALL |Rx|, Ry`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	`01011XXXYYY-----`
`CALL |Rx|, M`	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	`01100XXXMMMMMMMM`
`RET |Rx|`	Rx ← Rx+1 ; PC ← Mem[Rx]	`01101XXX--------`
`Libre`		`01110----------`
`Libre`		`01111----------`
`STR [M], Rx`	Mem[M] ← Rx	`10000XXXMMMMMMMM`
`LOAD Rx, [M]`	Rx ← Mem[M]	`10001XXXMMMMMMMM`
`STR [Rx], Ry`	Mem[Rx] ← Ry	`10010XXXYYY-----`
`LOAD Rx, [Ry]`	Rx ← Mem[Ry]	`10011XXXYYY-----`
`JMP M`	PC ← M	`10100---MMMMMMMM`
`JC M`	Si flag_C=1 entonces PC ← M	`10101---MMMMMMMM`
`JZ M`	Si flag_Z=1 entonces PC ← M	`10110---MMMMMMMM`
`JN M`	Si flag_N=1 entonces PC ← M	`10111---MMMMMMMM`
`JO M`	Si flag_O=1 entonces PC ← M	`11000---MMMMMMMM`
`SHRA Rx, t`	Rx ← Rx >>> t	`11001XXXYYY-----`
`SHR Rx, t`	Rx ← Rx >> t	`11010XXXYYY-----`
`SHL Rx, t`	Rx ← Rx << t	`11011XXXYYY-----`
`READF Rx`	Rx ← Flags	`11100XXX--------`
`LOADF Rx`	Flags ← Rx	`11101XXX--------`
`SET Rx, M`	Rx ← M	`11110XXXMMMMMMMM`
`Libre`		`11111----------`

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Conjunto de instrucciones

Decodificación

Instrucciones de 16 bits. Los primeros 5 bits identifican el opcode de la instrucción, el resto de los bits indican los parámetros.

Rx o Ry: Índices de registros, número entre 0 y 7.
M: Dirección de memoria o valor inmediato, número de 8 bits.
t: Valor inmediato de desplazamiento, número entre 0 y 7. Se codifica como YYY en 3 bits.

Ejemplos

Instrucción	Codificación
ADD R3, R6	0000101111000000
STR [0x21], R5	1000010100100001
SET R1, 0xFF	1111000111111111
SHL R7, 3	1101111101100000

Instrucción	Acción	Codificación
Reservada	Codifica el *Fetch* de instrucciones	00000----------
ADD Rx, Ry	Rx ← Rx + Ry	00001XXXYYY-----
ADC Rx, Ry	Rx ← Rx + Ry + Flag_C	00010XXXYYY-----
SUB Rx, Ry	Rx ← Rx - Ry	00011XXXYYY-----
AND Rx, Ry	Rx ← Rx and Ry	00100XXXYYY-----
OR Rx, Ry	Rx ← Rx or Ry	00101XXXYYY-----
XOR Rx, Ry	Rx ← Rx xor Ry	00110XXXYYY-----
CMP Rx, Ry	Modifica flags de Rx - Ry	00111XXXYYY-----
MOV Rx, Ry	Rx ← Ry	01000XXXYYY-----
PUSH |Rx|, Ry	Mem[Rx] ← Ry ; Rx ← Rx-1	01001XXXYYY-----
POP |Rx|, Ry	Rx ← Rx+1 ; Ry ← Mem[Rx]	01010XXXYYY-----
CALL |Rx|, Ry	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← Ry	01011XXXYYY-----
CALL |Rx|, M	Mem[Rx] ← PC ; Rx ← Rx-1 ; PC ← M	01100XXXMMMMMMMM
RET |Rx|	Rx ← Rx+1 ; PC ← Mem[Rx]	01101XXX--------
Libre		01110----------
Libre		01111----------
STR [M], Rx	Mem[M] ← Rx	10000XXXMMMMMMMM
LOAD Rx, [M]	Rx ← Mem[M]	10001XXXMMMMMMMM
STR [Rx], Ry	Mem[Rx] ← Ry	10010XXXYYY-----
LOAD Rx, [Ry]	Rx ← Mem[Ry]	10011XXXYYY-----
JMP M	PC ← M	10100---MMMMMMMM
JC M	Si flag_C=1 entonces PC ← M	10101---MMMMMMMM
JZ M	Si flag_Z=1 entonces PC ← M	10110---MMMMMMMM
JN M	Si flag_N=1 entonces PC ← M	10111---MMMMMMMM
JO M	Si flag_O=1 entonces PC ← M	11000---MMMMMMMM
SHRA Rx, t	Rx ← Rx >>> t	11001XXXYYY-----
SHR Rx, t	Rx ← Rx >> t	11010XXXYYY-----
SHL Rx, t	Rx ← Rx << t	11011XXXYYY-----
READF Rx	Rx ← Flags	11100XXX--------
LOADF Rx	Flags ← Rx	11101XXX--------
SET Rx, M	Rx ← M	11110XXXMMMMMMMM
Libre		11111----------

Los bits indicados con – deben valer cero.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
`00`	`SET R0, 0x01`	`R0 ← 01`

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R0, 0x01	R0 ← 01
02	SET R1, 0x05	R1 ← 05

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R0, 0x01	R0 ← 01
02	SET R1, 0x05	R1 ← 05
04	ADD R0, R1	R0 ← 05+01 = 06

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Nota:
`next` es una etiqueta, en ASM escribimos nombres que el ensablador convierte en direcciones de memoria.
`next` corresponde a la dirección de memoria `0x04`.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R0, 0x01	R0 ← 01
02	SET R1, 0x05	R1 ← 05
04	ADD R0, R1	R0 ← 05+01 = 06
06	JMP next	PC ← 04

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Nota:
`next` es una etiqueta, en ASM escribimos nombres que el ensablador convierte en direcciones de memoria.
`next` corresponde a la dirección de memoria `0x04`.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R0, 0x01	R0 ← 01
02	SET R1, 0x05	R1 ← 05
04	ADD R0, R1	R0 ← 05+01 = 06
06	JMP next	PC ← 04
04	ADD R0, R1	R0 ← 06+05 = 0B

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Sumas

Ensamblador	Memoria
`SET R0, 0x01`	`00 | f0 01`
`SET R1, 0x05`	`02 | f1 05`
`next: ADD R0, R1`	`04 | 08 20`
`JMP next`	`06 | a0 04`

Nota:
`next` es una etiqueta, en ASM escribimos nombres que el ensablador convierte en direcciones de memoria.
`next` corresponde a la dirección de memoria `0x04`.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R0, 0x01	R0 ← 01
02	SET R1, 0x05	R1 ← 05
04	ADD R0, R1	R0 ← 05+01 = 06
06	JMP next	PC ← 04
04	ADD R0, R1	R0 ← 06+05 = 0B
06	JMP next	PC ← 04
04	ADD R0, R1	R0 ← 0B+05 = 10
06	JMP next	PC ← 04
04	ADD R0, R1	R0 ← 10+05 = 15
06	JMP next	PC ← 04
04	ADD R0, R1	R0 ← 15+05 = 1A
...	...	...

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64
10	CMP R0, R3	N ← 0

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64
10	CMP R0, R3	N ← 0
12	JN maxR3	PC ← 14

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64
10	CMP R0, R3	N ← 0
12	JN maxR3	PC ← 14
14	JMP fin	PC ← 18

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64
10	CMP R0, R3	N ← 0
12	JN maxR3	PC ← 14
14	JMP fin	PC ← 18
18	STR [mayor], R0	MEM[1F] ← 64

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Ejemplo Mayor de 3

Ensamblador		Memoria
max3:	LOAD R1, [data1]	00 | 89 1c
	LOAD R2, [data2]	02 | 8a 1d
	LOAD R3, [data3]	04 | 8b 1e
	CMP R1, R2	06 | 39 40
	JN maxR2	08 | b8 0e
	MOV R0, R1	0a | 40 20
	JMP nextR3	0c | a0 10
maxR2:	MOV R0, R2	0e | 40 40
nextR3:	CMP R0, R3	10 | 38 60
	JN maxR3	12 | b8 16
	JMP fin	14 | a0 18
maxR3:	MOV R0, R3	16 | 40 60
fin:	STR [mayor], R0	18 | 80 1f
halt:	JMP halt	1a | a0 1a
data1:	DB 0x35	1c | 35
data2:	DB 0x64	1d | 64
data3:	DB 0x33	1e | 33
mayor:	DB 0x00	1f | 00

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	LOAD R1, [data1]	R1 ← 35
02	LOAD R2, [data2]	R2 ← 64
04	LOAD R3, [data3]	R3 ← 33
06	CMP R1, R2	N ← 1
08	JN maxR2	PC ← 0E
0e	MOV R0, R2	R0 ← 64
10	CMP R0, R3	N ← 0
12	JN maxR3	PC ← 14
14	JMP fin	PC ← 18
18	STR [mayor], R0	MEM[1F] ← 64
1a	JMP halt	PC ← 1A
...	...	...

Nota:
DB (Define Byte) es una meta instrucción que permite escribir bytes en la memoria.

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
`main:`	`SET R7, 0xFF`	`00 | f7 ff`
	`LOAD R0, [data]`	`02 | 88 1a`
	`CALL |R7|, f3n1`	`04 | 67 0a`
	`STR [data], R0`	`06 | 80 1a`
`halt:`	`JMP halt`	`08 | a0 08`
`f3n1:`	`PUSH |R7|, R1`	`0a | 4f 20`
	`MOV R1, R0`	`0c | 41 00`
	`ADD R0, R1`	`0e | 08 20`
	`ADD R0, R1`	`10 | 08 20`
	`SET R1, 1`	`12 | f1 01`
	`ADD R0, R1`	`14 | 08 20`
	`POP |R7|, R1`	`16 | 57 20`
	`RET |R7|`	`18 | 6f 00`
`data:`	`DB 0x23`	`1a | 23`

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
`00`	`SET R7, 0xFF`	`R7 ← FF`
`02`	`LOAD R0, [data]`	`R0 ← 23`

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

```
Ensamblador                    Memoria
main:  SET R7, 0xFF            00 | f7 ff
       LOAD R0, [data]         02 | 88 1a
       CALL |R7|, f3n1         04 | 67 0a
       STR [data], R0          06 | 80 1a
halt:  JMP halt                08 | a0 08
f3n1:  PUSH |R7|, R1           0a | 4f 20
       MOV R1, R0              0c | 41 00
       ADD R0, R1              0e | 08 20
       ADD R0, R1              10 | 08 20
       SET R1, 1               12 | f1 01
       ADD R0, R1              14 | 08 20
       POP |R7|, R1            16 | 57 20
       RET |R7|                18 | 6f 00
data:  DB 0x23                 1a | 23
```

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

	Ensamblador	Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A
16	POP |R7|, R1	R7←FE; R1←00

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A
16	POP |R7|, R1	R7←FE; R1←00
18	RET |R7|	R7←FF; PC←06

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A
16	POP |R7|, R1	R7←FE; R1←00
18	RET |R7|	R7←FF; PC←06
06	STR [data], R0	MEM[1a] ← 6A

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A
16	POP |R7|, R1	R7←FE; R1←00
18	RET |R7|	R7←FF; PC←06
06	STR [data], R0	MEM[1a] ← 6A
08	JMP halt	PC ← 08

Arquitectura OrgaSmall

Ejemplos en lenguaje ensamblador

Función $3 \cdot n + 1$

Ensamblador		Memoria
main:	SET R7, 0xFF	00 | f7 ff
	LOAD R0, [data]	02 | 88 1a
	CALL |R7|, f3n1	04 | 67 0a
	STR [data], R0	06 | 80 1a
halt:	JMP halt	08 | a0 08
f3n1:	PUSH |R7|, R1	0a | 4f 20
	MOV R1, R0	0c | 41 00
	ADD R0, R1	0e | 08 20
	ADD R0, R1	10 | 08 20
	SET R1, 1	12 | f1 01
	ADD R0, R1	14 | 08 20
	POP |R7|, R1	16 | 57 20
	RET |R7|	18 | 6f 00
data:	DB 0x23	1a | 23

Nota:
Las instrucciones CALL y RET permiten llamar a funciones.
Mientras que PUSH y POP salvar información en la pila.

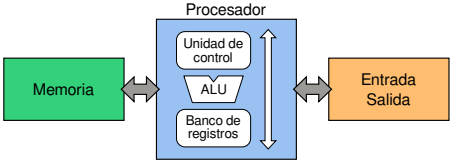
Seguimiento

Suponer inicialmente todos los registros en cero.

PC	Instrucción	Resultado
00	SET R7, 0xFF	R7 ← FF
02	LOAD R0, [data]	R0 ← 23
04	CALL |R7|, f3n1	MEM[FF]←06; R7←FE; PC←0a
0a	PUSH |R7|, R1	MEM[FE]←00; R7←FD
0c	MOV R1, R0	R1 ← 23
0e	ADD R0, R1	R0 ← 46
10	ADD R0, R1	R0 ← 69
12	SET R1, 1	R1 ← 1
14	ADD R0, R1	R0 ← 6A
16	POP |R7|, R1	R7←FE; R1←00
18	RET |R7|	R7←FF; PC←06
06	STR [data], R0	MEM[1a] ← 6A
08	JMP halt	PC ← 08
...	...	...

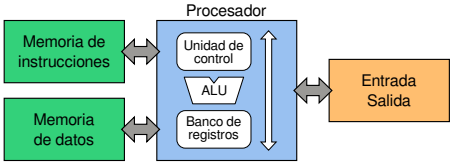
Arquitecturas

Arquitectura Von Neumann



- Consiste en tres sistemas de hardware: una **unidad central de procesamiento** CPU con una unidad de control, una ALU y registros. Una **memoria principal** que contiene datos y programas. Un subsistema de **entrada/salida**.
- Tiene la capacidad de ejecutar secuencialmente instrucciones. Contiene un único camino lógico entre la memoria principal y la unidad de control, forzando el acceso alternado entre datos e instrucciones. Este único camino es conocido como el **cuello de botella de von Neumann**.

Arquitectura Harvard



- Tienen **dos buses** para transferir simultáneamente datos e instrucciones desde **dos memorias independientes**.
- Las computadoras modernas utilizan una **versión modificada** teniendo buses separados para datos e instrucciones pero almacenamiento en una sola memoria.
- Las arquitecturas Harvard puras solo se utilizan en **microcontroladores** o sistemas integrados.

Conjunto de Instrucciones

Las instrucciones son la interfaz del procesador con el programador.
El *ISA* o *Instruction Set Architecture* especifica el conjunto de instrucciones que el procesador soporta y puede ejecutar.

Se pueden clasificar en tres grupos:

- **RISC** - *Reduced Instruction Set Computing*
  Arquitectura con una cantidad de instrucciones reducida. Simplifica el modelo de computo y ejecuta muy rápidamente las operaciones usando registros.
- **CISC** - *Complex Instruction Set Computing*
  Arquitectura con una gran cantidad de instrucciones, múltiples modos de direccionamiento y acceso a recursos.
- **MISC** - *Minimal Instruction Set Computing*
  Arquitectura con una cantidad limitada de instrucciones básicas. Estas incluso pueden estar limitadas a un propósito específico.

Taxonomía de Flynn

Nos permite clasificar instrucciones considerando dos factores.
El número de instrucciones a ejecutar y el número de datos sobre los cuales operar.

Generando las siguientes combinaciones:

- **SISD** - *single instruction stream, single data stream*
  Una sola instrucción opera sobre un solo dato.
- **SIMD** - *single instruction stream, multiple data streams*
  Una sola instrucción opera sobre múltiples datos.
- **MISD** - *multiple instruction streams, single data stream*
  Múltiples instrucciones operan simultáneamente sobre un solo dato.
- **MIMD** - *multiple instruction streams, multiple data streams*
  Múltiples instrucciones operan simultáneamente sobre un conjunto de datos.

Formato de instrucción

Para codificar un conjunto de instrucciones, debemos indicar qué valor binario tomará cada instrucción junto con sus parámetros.

Los formatos de instrucción pueden ser:

- De tamaño fijo
  Todas las instrucciones tienen el mismo tamaño.
- De tamaño variable
  Cada instrucción tiene un tamaño diferente, dependiendo de la operación a realizar y de sus parámetros.

Además, los formatos pueden **soportar extensiones**, es decir que algunas codificaciones de instrucciones estén libres para extender las operaciones que puede realizar el procesador.

Formato de instrucción - Intel

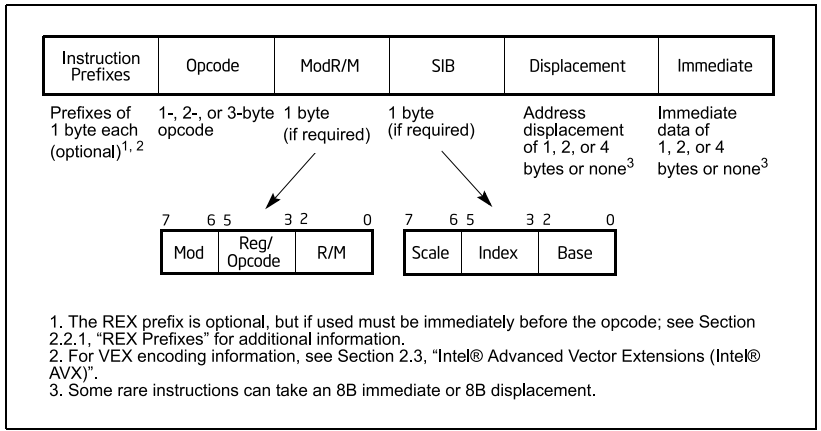


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Formato de instrucción - ARM

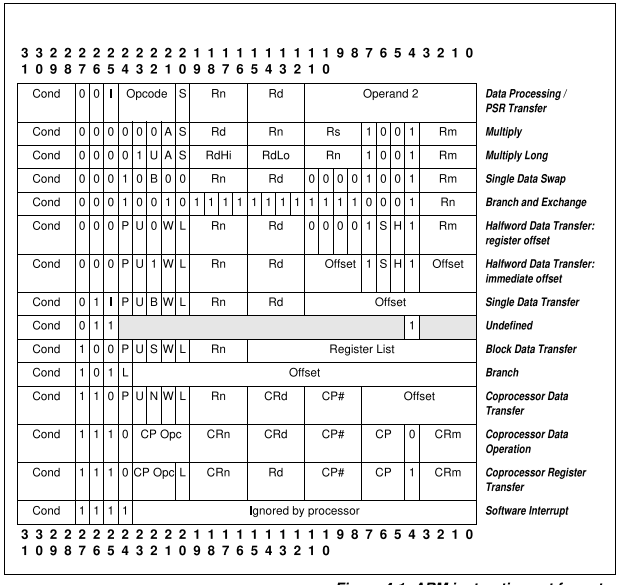


Figure 4-1: ARM instruction set formats

Modos de direccionamiento

Para ilustrar un conjunto de modos de direccionamiento vamos a utilizar la instrucción `MOV`.

- **Inmediato**

  `MOV R0, 0x1234` → `R0 = 0x1234`

  Mueve un valor inmediato codificado como parte de la instrucción a un registro.

Modos de direccionamiento

Para ilustrar un conjunto de modos de direccionamiento vamos a utilizar la instrucción `MOV`.

- **Inmediato**

  `MOV R0, 0x1234` → `R0 = 0x1234`

  Mueve un valor inmediato codificado como parte de la instrucción a un registro.

- **Directo**

  `MOV R0, R1` → `R0 = R1`

  Mueve el valor de un registro a otro registro.

Modos de direccionamiento

Para ilustrar un conjunto de modos de direccionamiento vamos a utilizar la instrucción `MOV`.

- **Inmediato**

  `MOV R0, 0x1234` → `R0 = 0x1234`

  Mueve un valor inmediato codificado como parte de la instrucción a un registro.

- **Directo**

  `MOV R0, R1` → `R0 = R1`

  Mueve el valor de un registro a otro registro.

- **Indirecto**

  `MOV R0, [0x1234]` → `R0 = MEM[0x1234]`

  Mueve el contenido de una dirección de memoria, codificada como parte de la instrucción, a un registro.

Modos de direccionamiento

Para ilustrar un conjunto de modos de direccionamiento vamos a utilizar la instrucción `MOV`.

- **Inmediato**

  `MOV R0, 0x1234` → `R0 = 0x1234`

  Mueve un valor inmediato codificado como parte de la instrucción a un registro.

- **Directo**

  `MOV R0, R1` → `R0 = R1`

  Mueve el valor de un registro a otro registro.

- **Indirecto**

  `MOV R0, [0x1234]` → `R0 = MEM[0x1234]`

  Mueve el contenido de una dirección de memoria, codificada como parte de la instrucción, a un registro.

- **Indirecto a registro**

  `MOV R0, [R1]` → `R0 = MEM[R1]`

  Mueve el contenido de una dirección de memoria apuntada por un registro a otro registro.

Modos de direccionamiento

Para ilustrar un conjunto de modos de direccionamiento vamos a utilizar la instrucción `MOV`.

- **Inmediato**

  `MOV R0, 0x1234` → `R0 = 0x1234`

  Mueve un valor inmediato codificado como parte de la instrucción a un registro.

- **Directo**

  `MOV R0, R1` → `R0 = R1`

  Mueve el valor de un registro a otro registro.

- **Indirecto**

  `MOV R0, [0x1234]` → `R0 = MEM[0x1234]`

  Mueve el contenido de una dirección de memoria, codificada como parte de la instrucción, a un registro.

- **Indirecto a registro**

  `MOV R0, [R1]` → `R0 = MEM[R1]`

  Mueve el contenido de una dirección de memoria apuntada por un registro a otro registro.

- **Indexado**

  `MOV R0, [R1 + 0x20]` → `R0 = MEM[R1+0x20]`

  Mueve el contenido de una dirección de memoria apuntada por un registro más un desplazamiento a otro registro.

big-endian vs little-endian

La memoria generalmente almacena datos de a **byte** (8 bits).
Se puede decir que funciona como un *gran* **arreglo de bytes**.

Por ejemplo, el número 1810 como entero de 16 bits (2 bytes):

```
0000011100010010b  →  0000 0111 0001 0010 b  →  07 12 h
```

big-endian vs little-endian

La memoria generalmente almacena datos de a **byte** (8 bits).
Se puede decir que funciona como un *gran* **arreglo de bytes**.

Por ejemplo, el número 1810 como entero de 16 bits (2 bytes):

`0000011100010010b → 0000 0111 0001 0010 b → 07 12 h`

Tenemos que guardar dos bytes en memoria,

- `big-endian`
  El número se almacena comenzando por el **byte más significativo**:

25	26	27	28	29	30	31	32
			07	12			

- `little-endian`
  El número se almacena comenzando por el **byte menos significativo**:

25	26	27	28	29	30	31	32
			12	07			

Bibliografía

- Tanenbaum, “Organización de Computadoras. Un Enfoque Estructurado”, 4ta Edición, 2000.
  - **Capítulo 5 - El nivel de Arquitectura del Conjunto de Instrucciones**
    - 5.3 Formatos de Instrucciones - Páginas 322 - 332
    - 5.4 Direccionamiento - Páginas 332 - 338

- Null, “Essentials of Computer Organization and Architecture”, 5th Edition, 2018.
  - Secciones:
    - 1.9 The Von Neumann Model
    - 1.10 Non-Von Neumann Models
    - 4.14 Real-Word Examples of Computer Architectures

Referencias

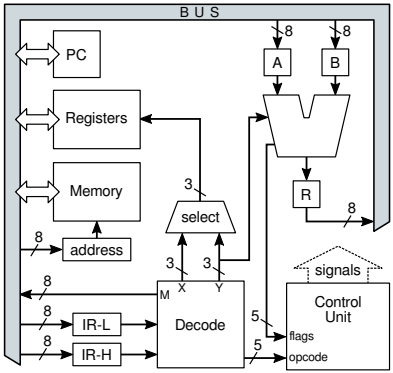
- Especificación de la Arquitectura OrgaSmall
  `https://github.com/fokerman/microOrgaSmall/`

¡Gracias!

Recuerden leer los comentarios adjuntos
en cada clase por aclaraciones.

Organización de OrgaSmall

La arquitectura *OrgaSmall* está completamente implementada en Logisim.



Sus componentes son:

- **RB** - `Registers` (Banco de Registros)
- **PC** - PC (Contador de Programa)
- **ALU** - ALU (Unidad Aritmético Lógica)
- **MM** - `Memory` (Memoria)
- **DE** - Decode (Decodificador de Instrucciones)
- **CU** - `ControlUnit` (Unidad de Control)

Todos están conectados a un bus común, y sus señales comandadas por la Unidad de Control.

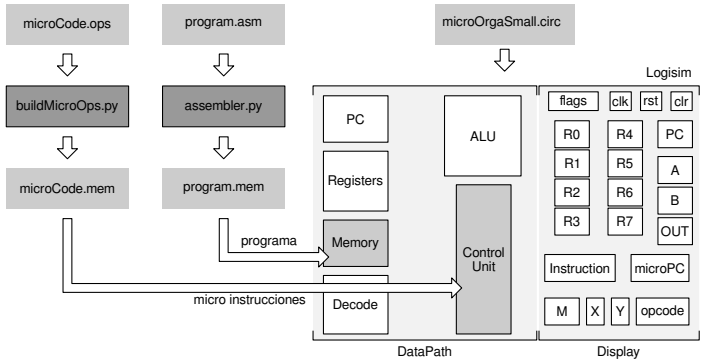
Organización de OrgaSmall

La unidad de control cuenta con las siguientes señales de entrada y salida:

Señales	**Descripción**
`inOpcode(5)`	Entrada de *Opcode*.
`flags(3)`	Entrada de *flags*.
`RB_enIn(1)` y `RB_enOut(1)`	Señales de habilitación para Registros.
`RB_inSelect(1)` y `RB_outSelect(1)`	Señales de selección para Registros.
`MM_enOut(1)` y `MM_load(1)`	Señales para la Memoria.
`MM_enAddr(1)`	Habilita cargar la dirección.
`ALU_enA(1)`, `ALU_enB(1)` y `ALU_enOut(1)`	Señales de habilitación de la ALU.
`ALU_opW(1)`, `ALU_OP(4)`	Señales de control de la ALU.
`PC_load(1)`, `PC_inc(1)` y `PC_enOut(1)`	Señales de control de PC.
`DE_enOutImm(1)`	Habilita la entrada al bus de un valor inmediato.
`DE_loadL(8)` y `DE_loadH(8)`	Indica cargar la mitad alta o baja.

Cada instrucción de 16 bits se decodifica a una secuencia de estas señales.

Herramientas para OrgaSmall



Para simplificar el desarrollo se proveen dos programas en python:

`assembler.py`

Toma un archivo en ensamblador de *OrgaSmall* (`.asm`) y genera un archivo (`.mem`) para cargar en la memoria de la máquina en Logisim.

`buildMicroOps.py`

Toma un archivo con micro-operaciones para la máquina *OrgaSmall* (`.ops`) y genera un archivo (`.mem`) para cargar en la memoria de micro-instrucciones de la máquina en Logisim.

Micro-instrucciones de OrgaSmall

El programa `buildMicroOps.py` toma un archivo con señales de la siguiente forma:

```
<binary_opcode>:
<signal_1> <signal_2> ... <signal_n>
...
<signal_1> <signal_2> ... <signal_n>
```

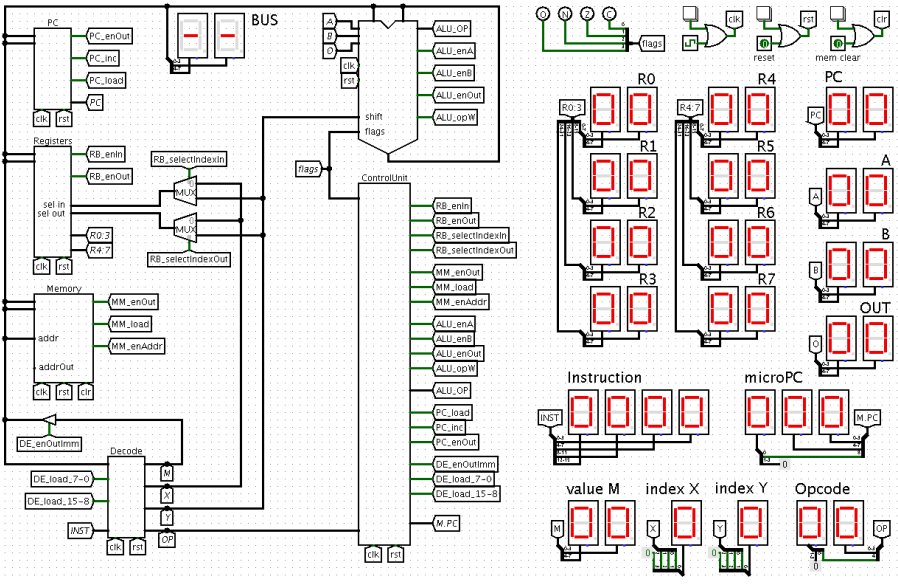
`binary_opcode`: Indica el Opcode de la instrucción a codificar.
`signal_x`: Corresponde a una señal en 1 (o `signal_x=val`).

Ejemplo para la instrucción ADD:

```
00001: ; ADD
    RB_enOut  ALU_enA  RB_selectIndexOut=0   ; ALU_A := Rx
    RB_enOut  ALU_enB  RB_selectIndexOut=1   ; ALU_B := Ry
    ALU_OP=ADD ALU_opW                       ; ALU_ADD
    RB_enIn   ALU_enOut RB_selectIndexIn=0   ; Rx := ALU_OUT
    reset_microOp
```

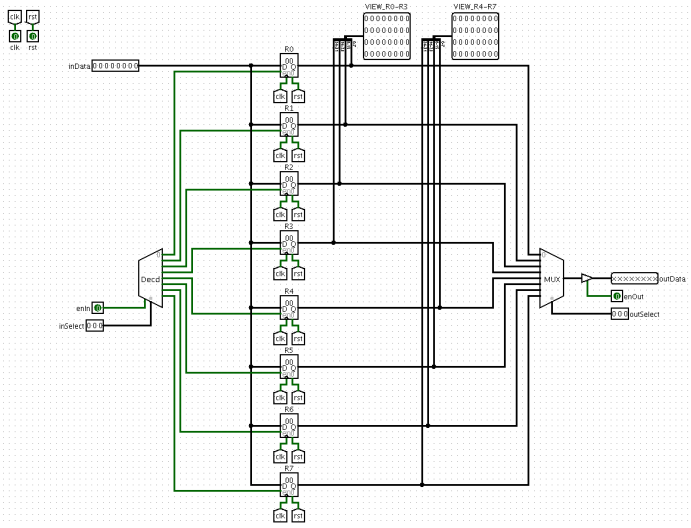
Además se puede utilizar el carácter ";" para indicar comentarios.

Organización de OrgaSmall



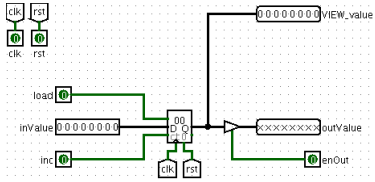
Organización de OrgaSmall

RB - `Registers` (Banco de Registros)



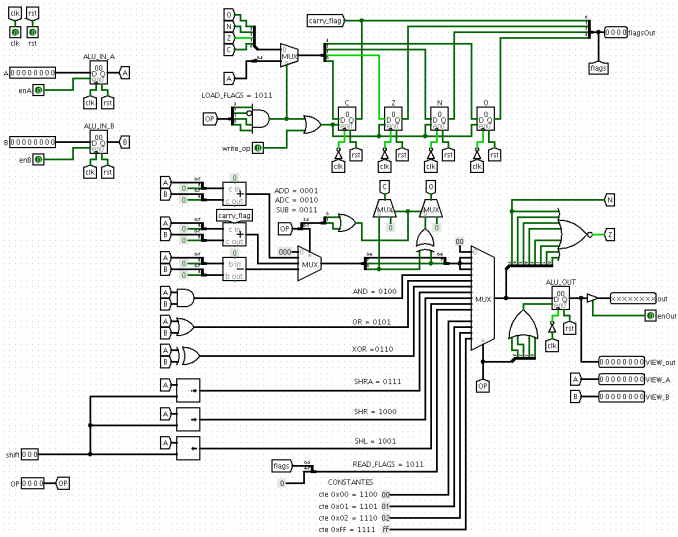
Organización de OrgaSmall

PC - PC (Contador de Programa)



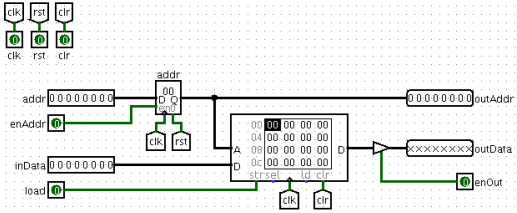
Organización de OrgaSmall

ALU - ALU (Unidad Aritmético Lógica)



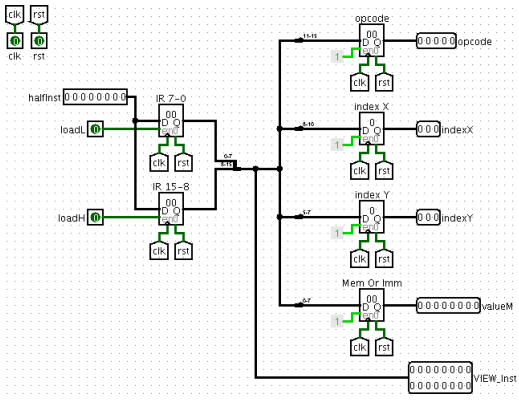
Organización de OrgaSmall

MM - Memory (Memoria)



Organización de OrgaSmall

DE - Decode (Decodificador de Instrucciones)



Organización de OrgaSmall

CU - `ControlUnit` (Unidad de Control)