Computer System Architecture

레지스터 전송과 마이크로 연산

- 레지스터 전송언어
- 레지스터 전송
- 버스와 메모리 전송
- 산술 마이크로 연산
- 논리 마이크로 연산
- 시프트 마이크로 연산
- 산술 논리 시프트 장치

간단한 디지털 시스템

- 간단한 디지털 시스템을 만들기위해 Combinational와 sequential circuits (learned in Chapters 1 and 2)이 사용된다.
- 이것들은 디지털 컴퓨터의 낮은 레벨을 구성한다.
- 간단한 디지털 시스템은 종종 다음의 특색을 이룬다...
  - the registers they contain, and
  - the operations that they perform.
- 대표적으로,
  - What operations are performed on the data in the registers
  - What information is passed between registers

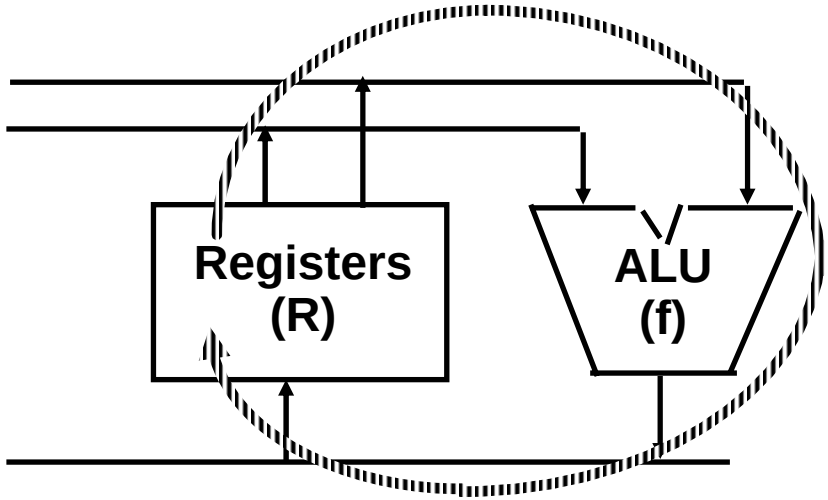
마이크로 연산 (1)

- 레지스터안의 데이터들로 연산되는 것을 마이크로 연산이라한다.

- 레지스터안에서 수행되는 연산, 마이크로 연산의 예의다.
  - Shift
  - Load
  - Clear
  - Increment
  - ...

마이크로 연산 (2)

An elementary operation performed (during one clock pulse), on the information stored in one or more registers



1 clock cycle

$R \leftarrow f(R, R)$

f: shift, load, clear, increment, add, subtract, complement, and, or, xor, …

디지털 시스템의 구성

❖ 컴퓨터 내부구성의 기술

- 레지스터의 셋팅과 그들의 기능.
- **Micro operations set**
  컴퓨터 구성의 한계내에서 하이크로 연산이 허용된다.
- 마이크로 연산을 연속적으로 수행함으로 신호를 제어 한다.
  (기능수행을 위해서**...)**

레지스터 전송 레벨

- **컴퓨터처럼 보이는 디지털 시스템들은 데이터 전송레벨이라 불리는 방법을 가지고 있다.**

- **중요 관점.**
  - The system's registers
  - The data transformations in them, and
  - The data transfers between them.

레지스터 전송 언어

- 레지스터 전송언어는 더 낮은 디지털 시스템 표현으로 일정한 표시가 사용된다.
- 컴퓨터의 연산을 위해서 레지스터 전송언어로 표현된 것을 사용한다.
- 레지스터 전송 언어
  - 상징적인 언어
  - 디지털 컴퓨터의 내부 구성을 기술하는 편리한 툴이다.
  - 디지털 시스템 디자인 과정을 쉽게 하는데 이용된다.

레지스터 디자인

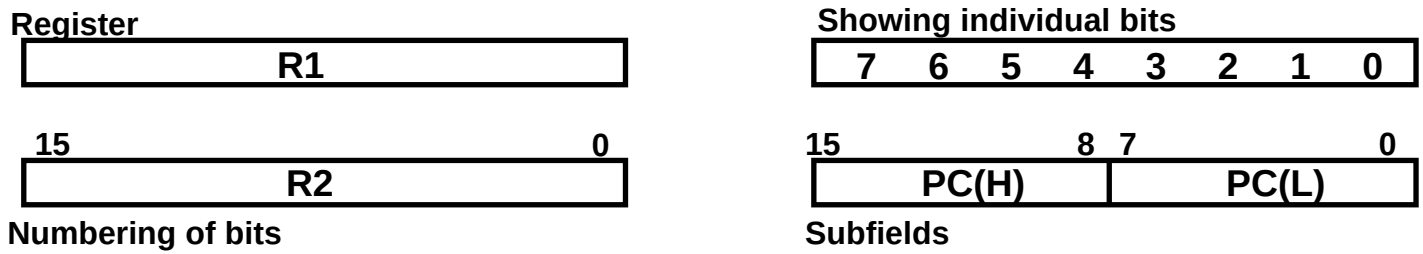
- 레지스터는 대문자로 표현된다. 종종 숫자에 따라 구분되기도 한다. (e.g., A, R13, IR)
- 자주 쓰이는 기능 표현:
  - MAR - memory address register
  - PC - program counter
  - IR - instruction register
- 레지스터와 그 내용은 많은 많은 방법들로 묘사되어 보여 진다.
  - A register can be viewed as a single entity:

  - Registers may also be represented showing the bits of data they contain

레지스터 디자인

- **Designation of a register**
  - **a register**
  - **portion of a register**
  - **a bit of a register**

- **Common ways of drawing the block diagram of a register**



레지스터 전송

- 어떤 레지스터로 다른 레지스터를 복사하는것
- 레지스터 전송의 바람직한 결과

R2 ← R1

  - 이 경우 R1의 내용이 R2에 복사된다.
  - 1클럭펄스 동안에 R1에서 R2로 모든 비트의 전송이 이루어진다.
  - 주의 해야 될점은 데이터가 보존 되어야 된다는점이다
    ; i.e. 에서 R1의 내용이 R2로 변환되면서 바뀌지 않는다고 정의 한다.

레지스터 전송

- A register transfer such as

  R3 ← R5

  디지털 시스템에서는 다음을 의미 한다.

  - **데이터 선은 전송레지스터**(source register)(R5)**와 목적지 레지스터**(destination register (R3))**로 이어 진다.**
  - R3**는 목적지로부터 병렬로** load**되어 진다.**
  - **동작 수행은 제어 라인으로 연결 한다.**

제어 기능 (*control function*)

- 조건이 만족 할때 실행 되어야 한다.
- 이것을 프로 그래밍언어에서 “if”문법과 비슷하다.
- 디지털 시스템에서는 , 제어 신호를 거쳐 자구 일어나는데 이를 제어 기능 (*control function*)이라 한다.
  - 신호가 1이라면, 특정 장소에서 수행 된다.
- 묘사를 한다면…

P: R2 ← R1

만약 P = 1이라면, R1의 내용을 R2안으로 읽어오는 것이다.”,

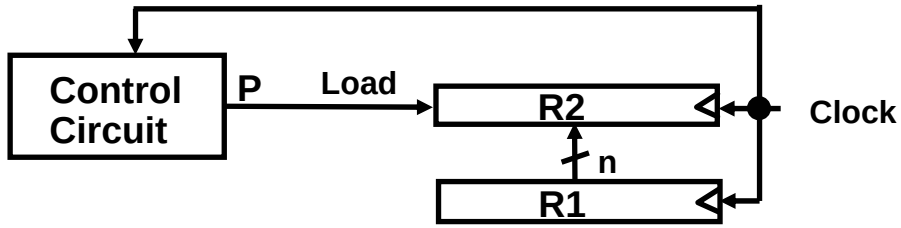
i.e., if (P = 1) then (R2 ← R1)

HARDWARE IMPLEMENTATION OF CONTROLLED TRANSFERS
전송 콘트롤의 하드웨어적 수행

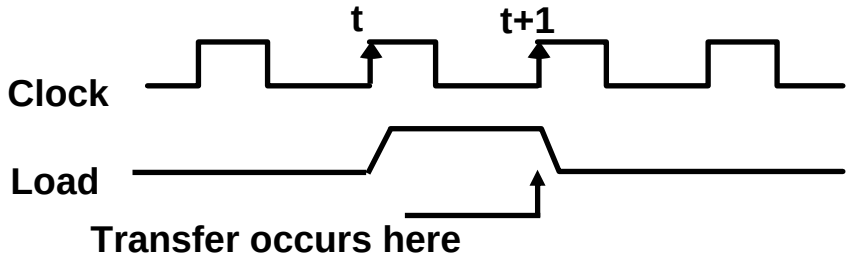
Implementation of

P: R2 ← R1

Block diagram



Timing diagram



- **The same clock controls the circuits that generate the control function and the destination register**
- **Registers are assumed to use *positive-edge-triggered* flip-flops**

동시 수행

- 만약 두개의 명령이 동시에 수행된다면, 콤마(,)로 나누어 나타낸다.

  P: R3 ← R5, MAR ← IR

- 만약 콘트롤 제어가 P = 1이면, R5을 R3으로 Load하고, 같은 클럭 시간에 (clock), 레지스터를 MAR로 불러 온다.

BASIC SYMBOLS FOR REGISTER TRANSFERS

Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A ← B, B ← A

레지스터 접속

- 디지털시스템에서는 많은 레시스터를 가지고 있다.
- 레지스터에는 가상적인 데이터 값이나 제어 입력을 가지고 있어 직접적으로 다른 레지스터를 읽거나 다른레지스터 내용을 참조하여 레지스터에 접근한다.

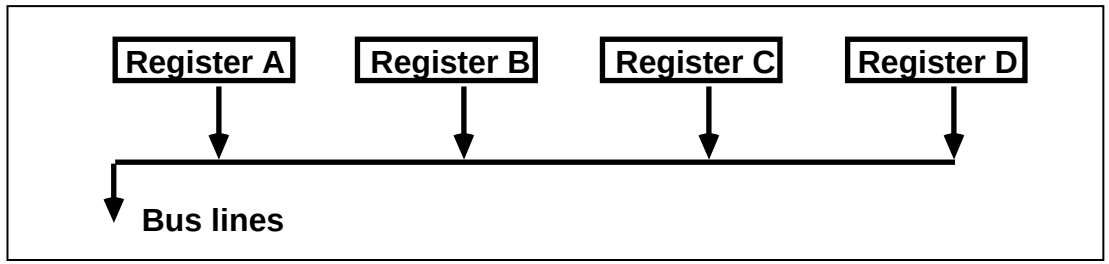
- 완전히 n레지스터에 접근 → n(n-1) lines
- $O(n^2)$ cost
  - 큰 디지털 시스템에서는 사실적으로 접근하여 사용하지 않는다.

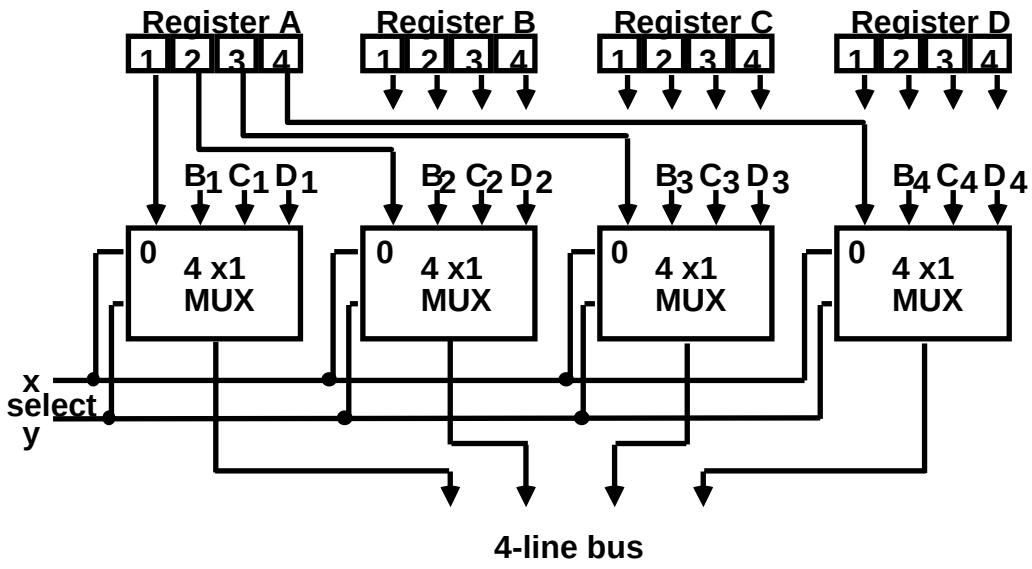
  (This is not a realistic approach to use in a large digital system.)
  - 대신 다르게 접근한다.
- 회로 전송을 위해 하나로 모은다. – the bus
- 제어의 선탣을 위해 레지스터의 수신지나 목적지를 설정한다.

버스와 버스 전송

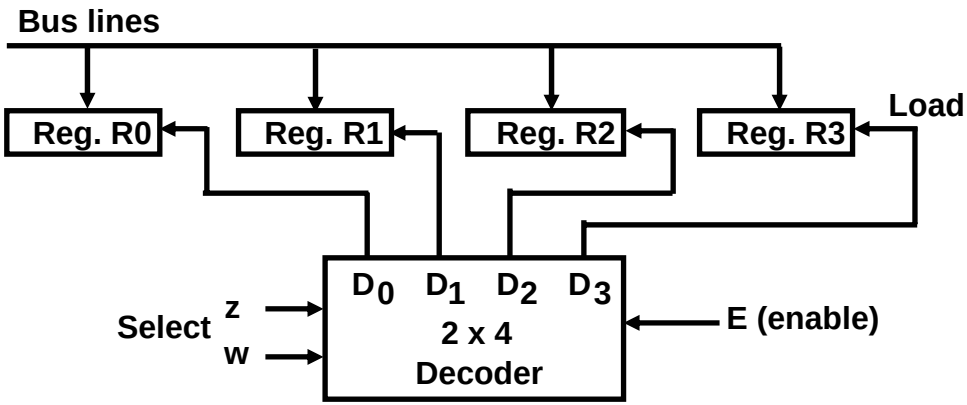
Bus is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

From a register to bus: BUS ← R





목적지 레지스터의 버스의 전송 구조



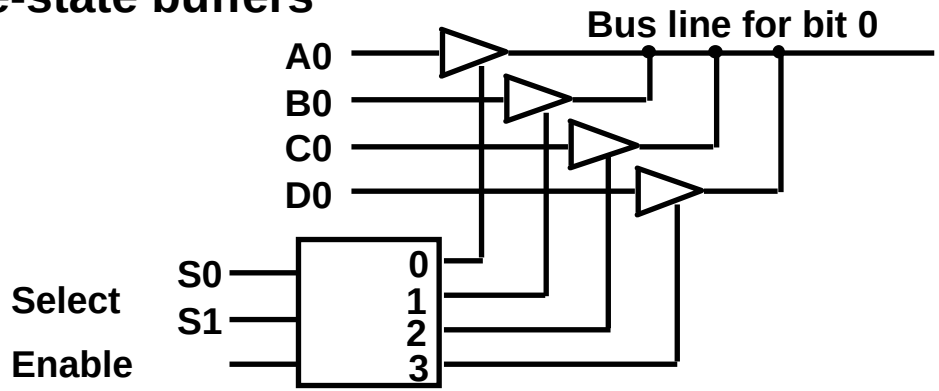
Three-State Bus Buffers

Normal input A

Control input C

Output Y=A if C=1
High-impedence if C=0

Bus line with three-state buffers



RTL에서의 버스 전송

- 레지스터의 전송은 버스가 명확히 언급되었는가 아닌가에 따라서 명백히 달라진다.

$$R2 \leftarrow R1$$

or

$$BUS \leftarrow R1,\ R2 \leftarrow BUS$$

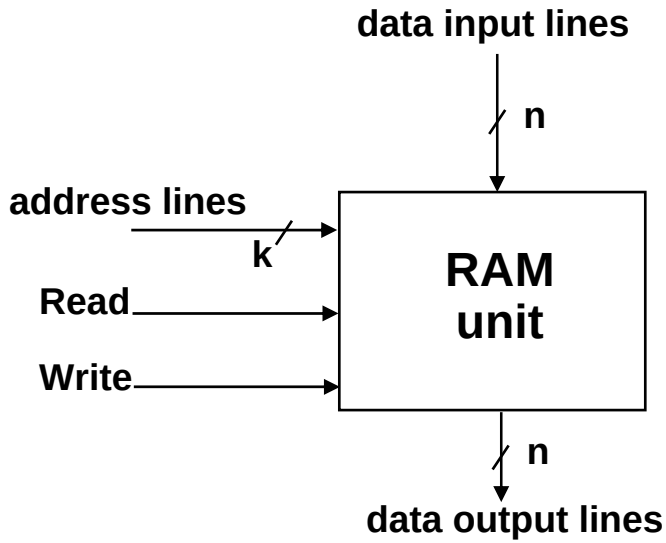
- 앞의 것은 버스의 은연중의 실행이다.
- 뒤의 것은 바람직한 실행이다.

MEMORY (RAM)

- 메모리는 레지스터의 숫자를 포함하는 순환 회로에 접근한다.
- 레지스터는 메모리의 워드 정보를 가지고 있다.
- R레지스터는 주소 정보를 가지고 있다.
- 주소 범위는 0 to r-1
- 각각 레지스터는 데이터의 n비트 가지고 있다.
- Assume the RAM contains r = 2k words.

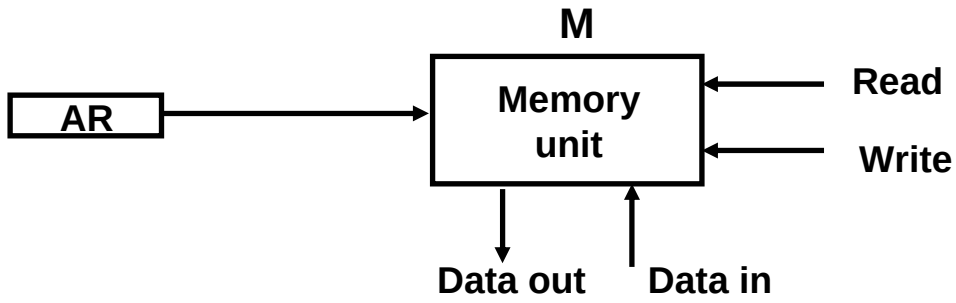
= 이것은 다음에 필요하다. =

- n data input lines
- n data output lines
- k address lines
- A Read control line
- A Write control line



MEMORY TRANSFER

- 공통적으로, 메모리는 레지스터 장치 레벨에서 M으로 나타내어 진다.
- 여러 장소에서의 사용중 이라면 우리가 사용하는 메모리의 내용을 표시할 것이다.
- 이것은 메모리참조 표기에 쓰인다.
- 메모리는 컴퓨터 시스템에서 특별한 레지스터 에 접근할 때 쓰인다. 그 메모리는 (MAR,AR)에 저장된다.
- 메모리에 접근할 때, MAR의 내용을 접근하여 특별한 메모리 어드레스 라인에 보낸다.



MEMORY READ

- 메모리 저장소와 레지스터 안의 내용, 레지스터 전송 표기 값을 읽기는 표현은 다음과 같다.:

$$R1 \leftarrow M[MAR]$$

- 실행 결과 분석.
  - The contents of the MAR get sent to the memory address lines
  - A Read (= 1) gets sent to the memory unit
  - The contents of the specified address are put on the memory's output data lines
  - These get sent over the bus to be loaded into register R1

MEMORY WRITE

- 메모리 저장소와 레지스터 안의 내용, 레지스터 전송 표기 값을 쓰는 표현은 다음과 같다.:

$$M[MAR] \leftarrow R1$$

- 실행 결과 분석.
  - The contents of the MAR get sent to the memory address lines
  - A Write (= 1) gets sent to the memory unit
  - The values in register R1 get sent over the bus to the data input lines of the memory
  - The values get loaded into the specified address in the memory

SUMMARY OF R. TRANSFER MICROOPERATIONS

A ← B	Transfer content of reg. B into reg. A
AR ← DR(AD)	Transfer content of AD portion of reg. DR into reg. AR
A ← constant	Transfer a binary constant into reg. A
ABUS ← R1,	Transfer content of R1 into bus A and, at the same time,
R2 ← ABUS	transfer content of bus A into R2
AR	Address register
DR	Data register
M[R]	Memory word specified by reg. R
M	Equivalent to M[AR]
DR ← M	Memory *read* operation: transfers content of memory word specified by AR into DR
M ← DR	Memory *write* operation: transfers content of DR into memory word specified by AR

마이크로 연산

❖ 컴퓨터 시스템에서 마이크로 연산은 4가지 타입이 있다.

- 레지스터 전송 마이크로 연산.
- 산술 마이크로 연산
- 논리 마이크로 연산
- 시프트 마이크로 연산

산술 마이크로 연산

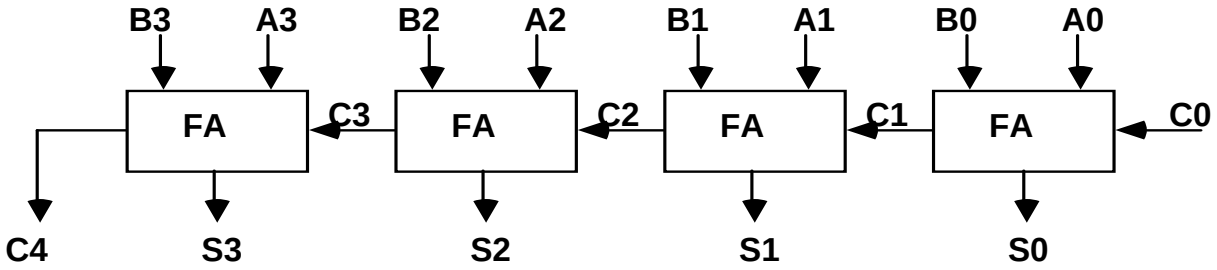
- **기본적인 산술 마이크로 연산**
  - Addition
  - Subtraction
  - Increment
  - Decrement
- **추가적인 산술 마이크로 연산**
  - Add with carry
  - Subtract with borrow
  - Transfer/Load
  - etc. ...

Summary of Typical Arithmetic Micro-Operations

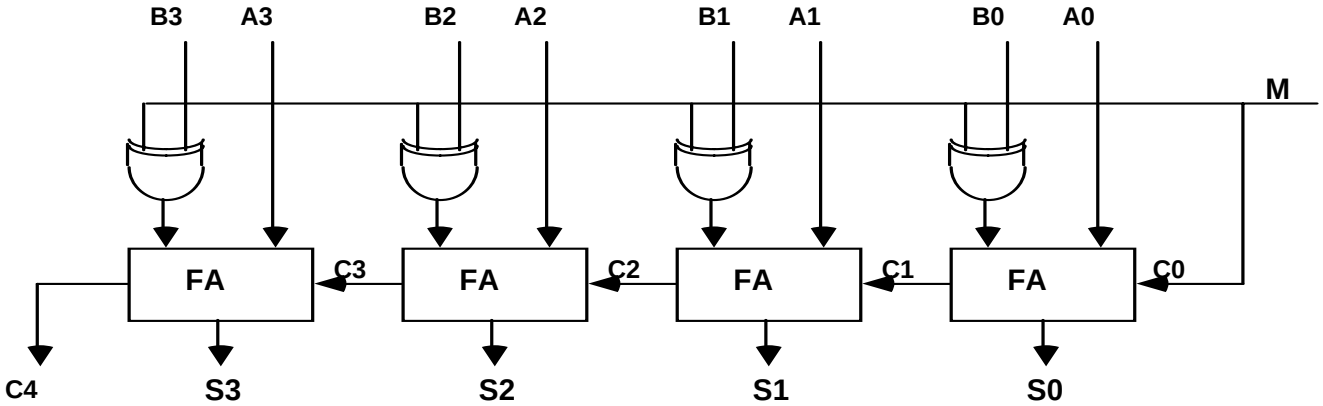
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

BINARY ADDER / SUBTRACTOR / INCREMENTER

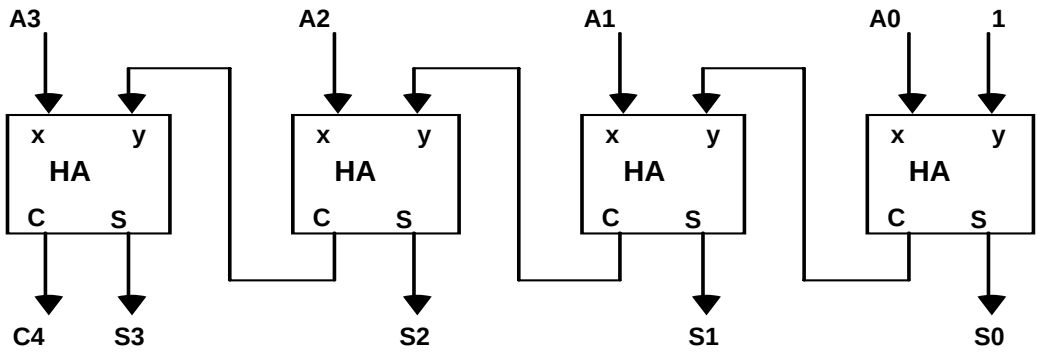
Binary Adder



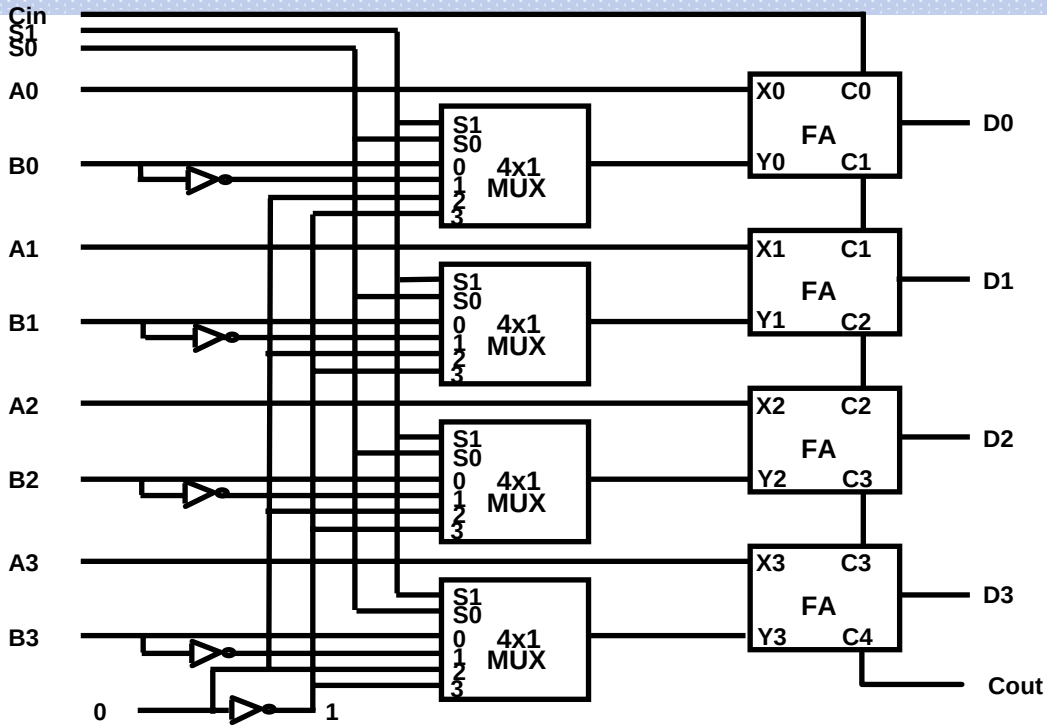
Binary Adder-Subtractor



Binary Incrementer



산술 회로



S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B’	D = A + B’	Subtract with borrow
0	1	1	B’	D = A + B’+ 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A - 1	Decrement A
1	1	1	1	D = A	Transfer A

논리 마이크로 연산

- Specify binary operations on the strings of bits in registers
  - Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data
  - useful for bit manipulations on binary data
  - useful for making logical decisions based on the bit value
- There are, in principle, 16 different logic functions that can be defined over two binary input variables

A	B	F_0	F_1	F_2	…	F_{13}	F_{14}	F_{15}
0	0	0	0	0	…	1	1	1
0	1	0	0	0	…	1	1	1
1	0	0	0	1	…	0	1	1
1	1	0	1	0	…	1	0	1

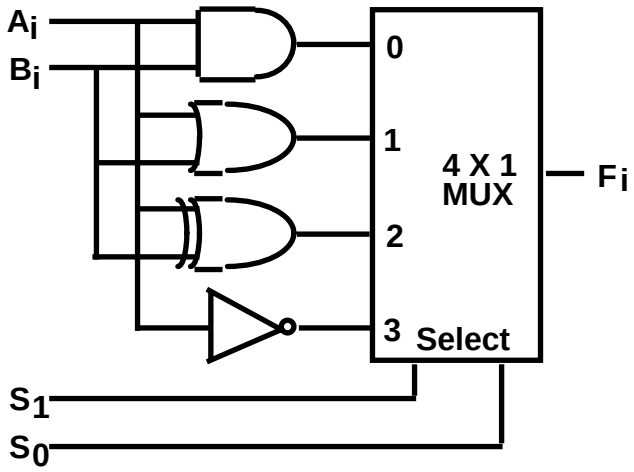
- However, most systems only implement four of these
  - AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), Complement/NOT
- The others can be created from combination of these

논리 마이크로 연산 리스트

- 논리 마이크로 연산 리스트
  - **16 different logic operations with 2 binary vars.**
  - **n binary vars → 2^{2^n} functions**
- **2**개의 값과 대응하는 **16** 마이크로 연산 로직 진리표.

x y	0 0 1 1 0 1 0 1	*Boolean Function*	*Micro-Operations*	*Name*
	0 0 0 0	F0 = 0	F ← 0	Clear
	0 0 0 1	F1 = xy	F ← A ∧ B	AND
	0 0 1 0	F2 = xy'	F ← A ∧ B’	
	0 0 1 1	F3 = x	F ← A	Transfer A
	0 1 0 0	F4 = x'y	F ← A’∧ B	
	0 1 0 1	F5 = y	F ← B	Transfer B
	0 1 1 0	F6 = x ⊕ y	F ← A ⊕ B	Exclusive-OR
	0 1 1 1	F7 = x + y	F ← A ∨ B	OR
	1 0 0 0	F8 = (x + y)'	F ← (A ∨ B)’	NOR
	1 0 0 1	F9 = (x ⊕ y)'	F ← (A ⊕ B)’	Exclusive-NOR
	1 0 1 0	F10 = y'	F ← B’	Complement B
	1 0 1 1	F11 = x + y'	F ← A ∨ B	
	1 1 0 0	F12 = x'	F ← A’	Complement A
	1 1 0 1	F13 = x' + y	F ← A’∨ B	
	1 1 1 0	F14 = (xy)'	F ← (A ∧ B)’	NAND
	1 1 1 1	F15 = 1	F ← all 1's	Set to all 1's

논리 마이크로 연산 의 하드웨어 구성



Function table

S_1	S_0	Output	μ-operation
0	0	F = A ∧ B	AND
0	1	F = A ∨ B	OR
1	0	F = A ⊕ B	XOR
1	1	F = A'	Complement

논리 마이크로 연산 적용하기

- 논리 마이트로 연산은 레지트터 안의 명령어 각각의 비트를 조정한다.
- A레지스터안의 데이터를 참조한다. 다른 레지스터 B 는 A에 만족하게 수정한다.

Selective-set	$A \leftarrow A + B$
Selective-complement	$A \leftarrow A \oplus B$
Selective-clear	$A \leftarrow A \bullet B'$
Mask (Delete)	$A \leftarrow A \bullet B$
Clear	$A \leftarrow A \oplus B$
Insert	$A \leftarrow (A \bullet B) + C$
Compare	$A \leftarrow A \oplus B$
. . .	

SELECTIVE SET

- In a selective set operation, the bit pattern in B is used to *set* certain bits in A

$$\begin{array}{ll} 1\ 1\ 0\ 0\ A_t & \\ 1\ 0\ 1\ 0\ B & \\ \hline 1\ 1\ 1\ 0\ A_{t+1} & (A \leftarrow A + B) \end{array}$$

- If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value

SELECTIVE 보완

- In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A

$$\begin{array}{ll} 1\ 1\ 0\ 0\ A_t & \\ 1\ 0\ 1\ 0\ B & \\ \hline 0\ 1\ 1\ 0\ A_{t+1} & (A \leftarrow A \oplus B) \end{array}$$

- If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged

CLEAR 선택

- In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A

$$
\begin{array}{ll}
1\ 1\ 0\ 0\ A_t & \\
1\ 0\ 1\ 0\ B & \\
\hline
0\ 1\ 0\ 0\ A_{t+1} & (A \leftarrow A \cdot B')
\end{array}
$$

- If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged

MASK 연산

- In a mask operation, the bit pattern in B is used to *clear* certain bits in A

$$
\begin{array}{ll}
1\ 1\ 0\ 0\ A_t & \\
1\ 0\ 1\ 0\ B & \\
\hline
1\ 0\ 0\ 0\ A_{t+1} & (A \leftarrow A \cdot B)
\end{array}
$$

- If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged

CLEAR 연산

- In a clear operation, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A

$$
\begin{array}{ll}
1\ 1\ 0\ 0\ A_t & \\
1\ 0\ 1\ 0\ B & \\
\hline
0\ 1\ 1\ 0\ A_{t+1} & (A \leftarrow A \oplus B)
\end{array}
$$

연산 삽입

- An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged
- This is done as
  - A mask operation to clear the desired bit positions, followed by
  - An OR operation to introduce the new bits into the desired positions
  - Example
    - Suppose you wanted to introduce 1010 into the low order four bits of A:

      1101 1000 1011 0001 A (Original)

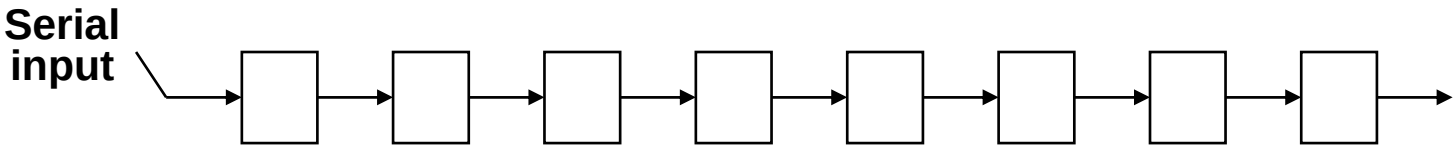
      1101 1000 1011 1010 A (Desired)

    -
```
1101 1000 1011 0001          A (Original)
1111 1111 1111 0000          Mask
1101 1000 1011 0000          A
(Intermediate)
0000 0000 0000 1010          Added bits
1101 1000 1011 1010          A (Desired)
```

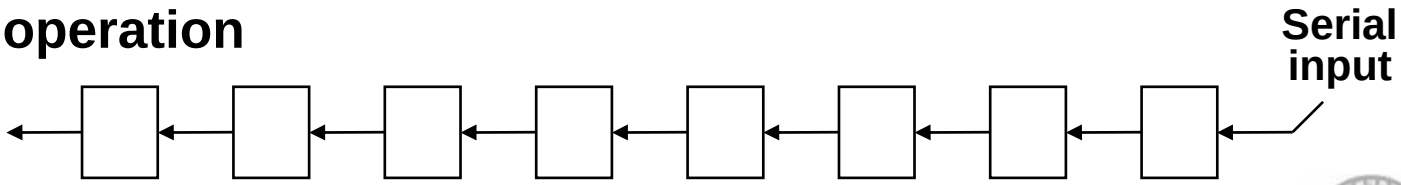
시프트 마이크로 연산

- There are three types of shifts
  - *Logical shift*
  - *Circular shift*
  - *Arithmetic shift*
- What differentiates them is the information that goes into the serial input

• **A right shift operation**

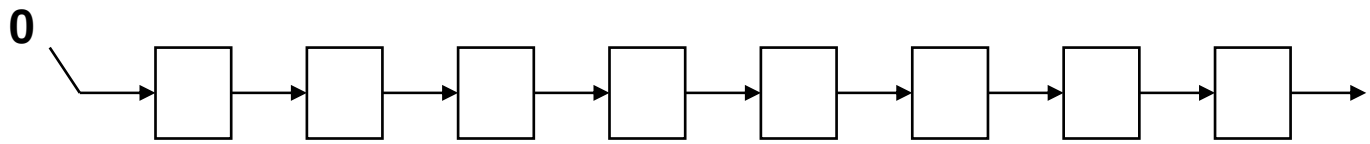


• **A left shift operation**

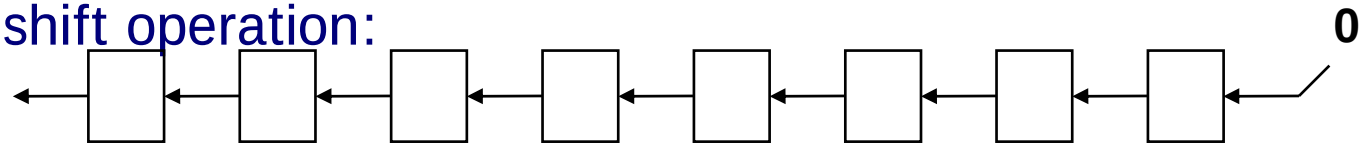


논리 시프트

- In a logical shift the serial input to the shift is a 0.
- A right logical shift operation:



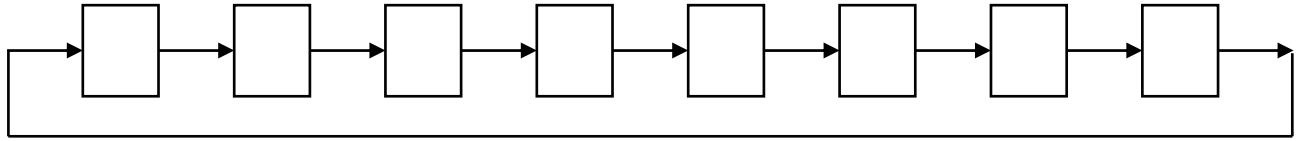
- A left logical shift operation:



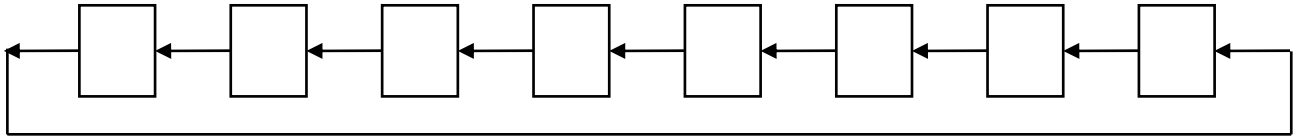
- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right
  - Examples:
    - R2 ← *shr* R2
    - R3 ← *shl* R3

순환 시프트

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.
- A right circular shift operation:



- A left circular shift operation:

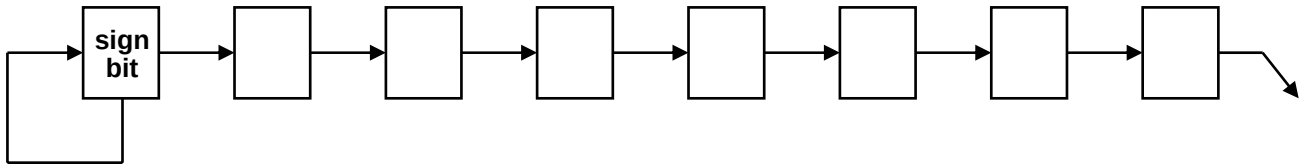


- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right
  - Examples:
    - R2 ← *cir* R2
    - R3 ← *cil* R3

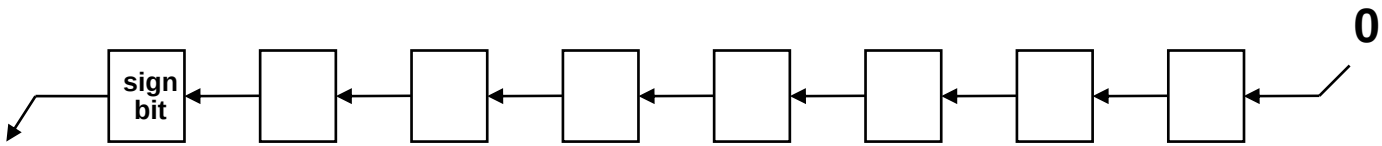
산술 시프트

- 산술시프트는 정수 바이너리 표현을 의미한다. (integer)
- 왼쪽 시프트는 2를 곱하는 연산과 같다.
- 오른 시프트는 2를 나누는 연산과 같다.
- 산술시프트에서 중요한 점은 실행시 수의 부호를 지키면서 곱하기나 나누기 연산을 실행하는 것이다.

- A right arithmetic shift operation:

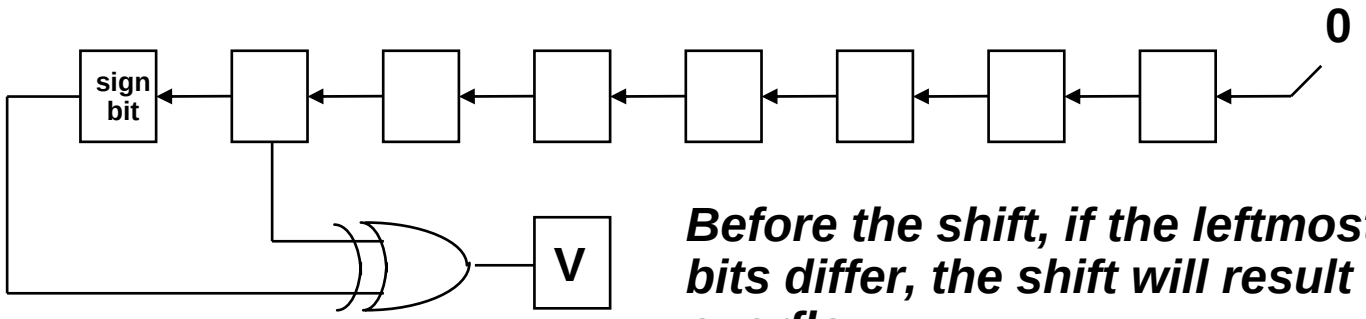


- A left arithmetic shift operation:



산술 시프트

❖ 왼쪽 산술 연산에서는 overflow를 체크 하여야 한다.

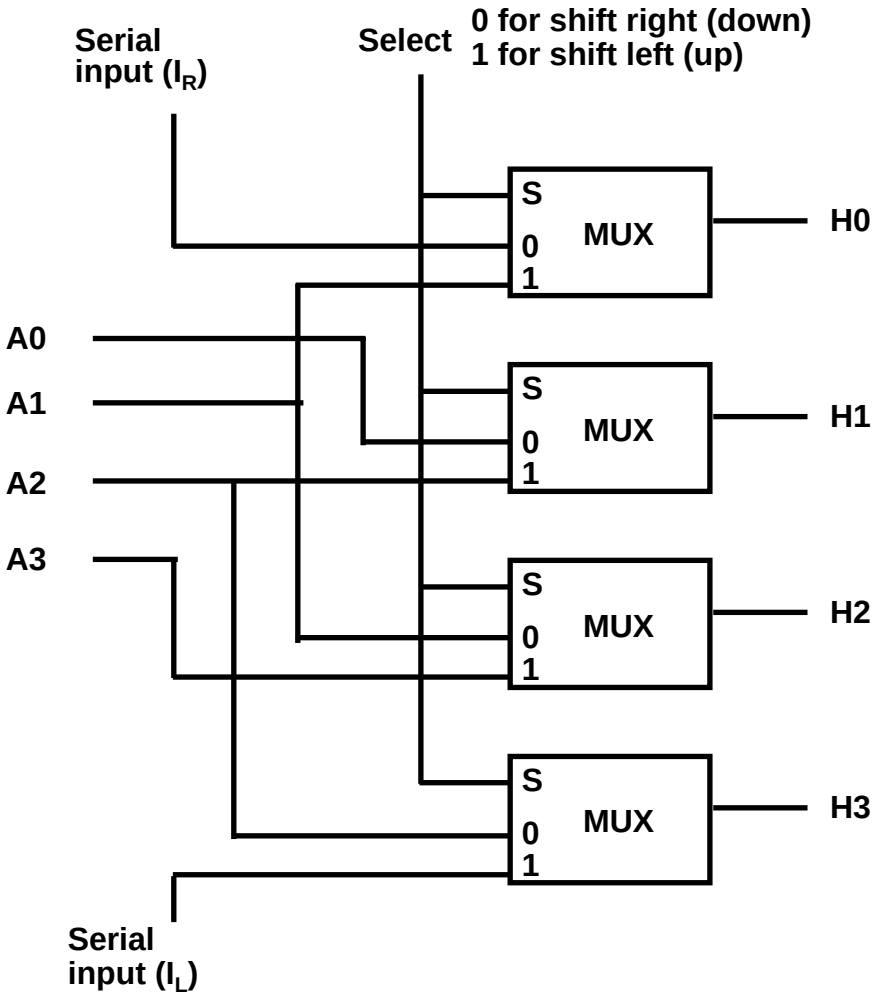


Before the shift, if the leftmost two bits differ, the shift will result in an overflow

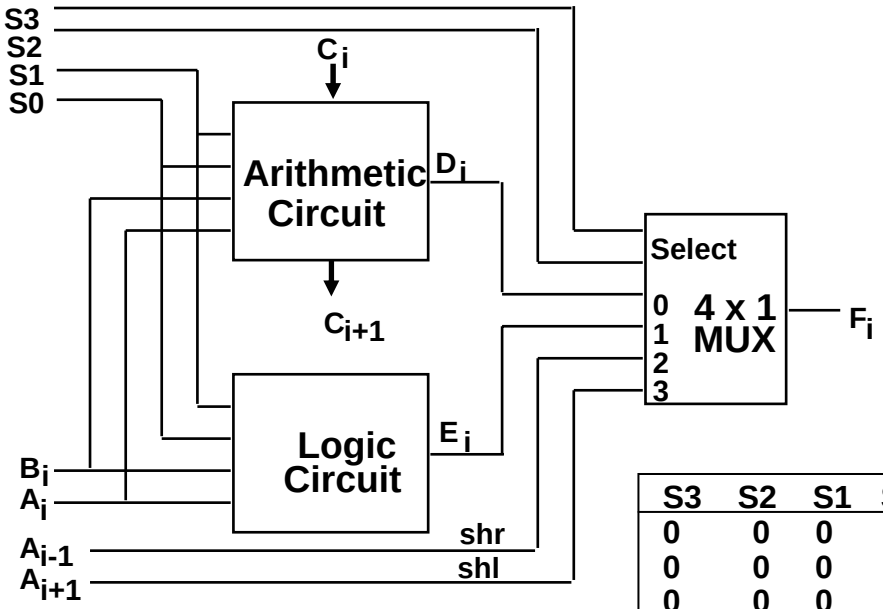
❖ RTL에서는 다음의 표기 법을 사용한다.

- *ashl* for an arithmetic shift left
- *ashr* for an arithmetic shift right
- Examples:
  - R2 ← *ashr* R2
  - R3 ← *ashl* R3

시프트 마이크로 연산의 하드웨어 구현



산술 논리 시프트 장치



S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B'+ 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = A'	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F