그림 1

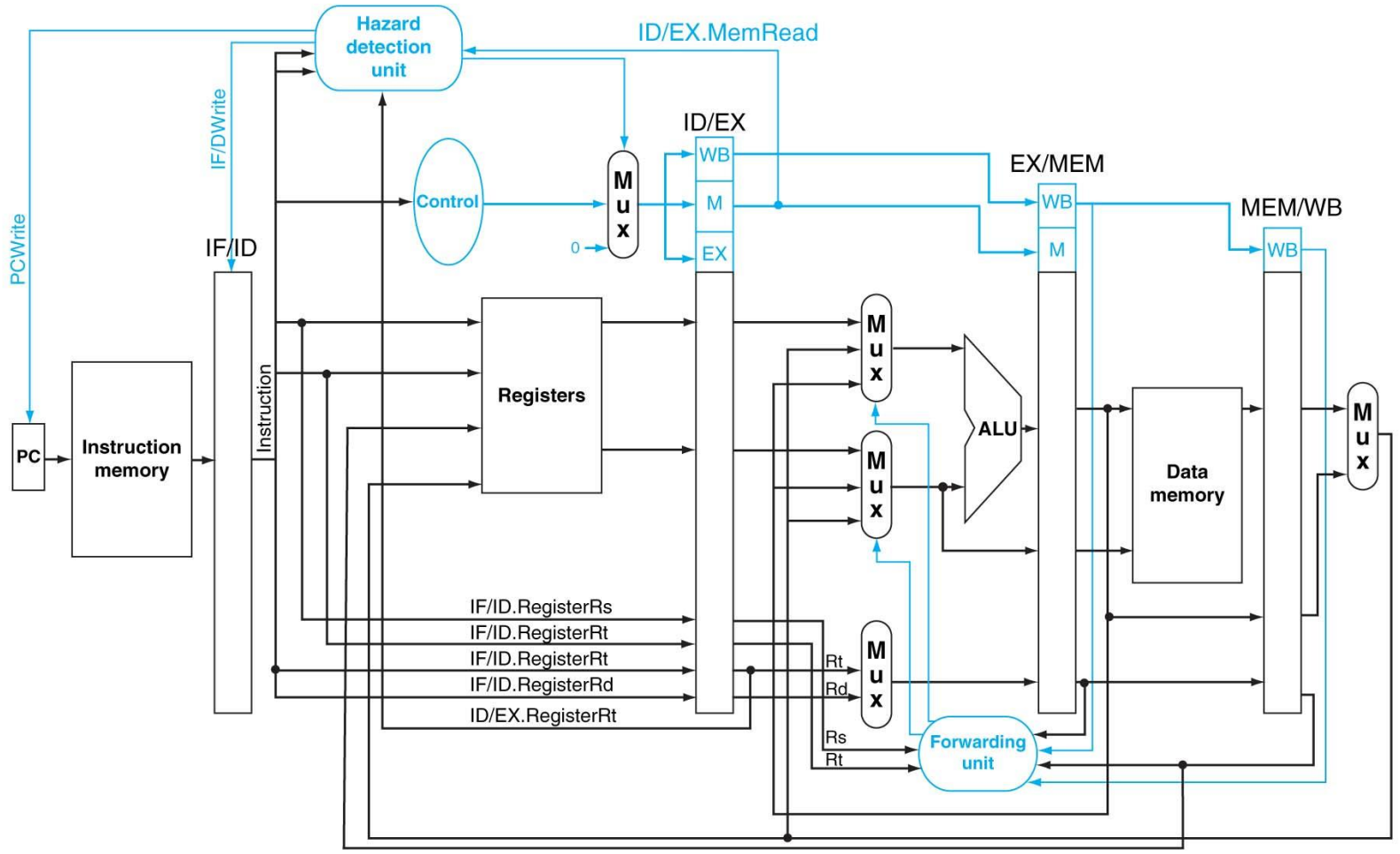

Hazard detection unit
ID/EX.MemRead
IF/DWrite
PCWrite
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
Mux
0
IF/ID
Instruction
PC
Instruction memory
Registers
ALU
Data memory
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit

그림 2

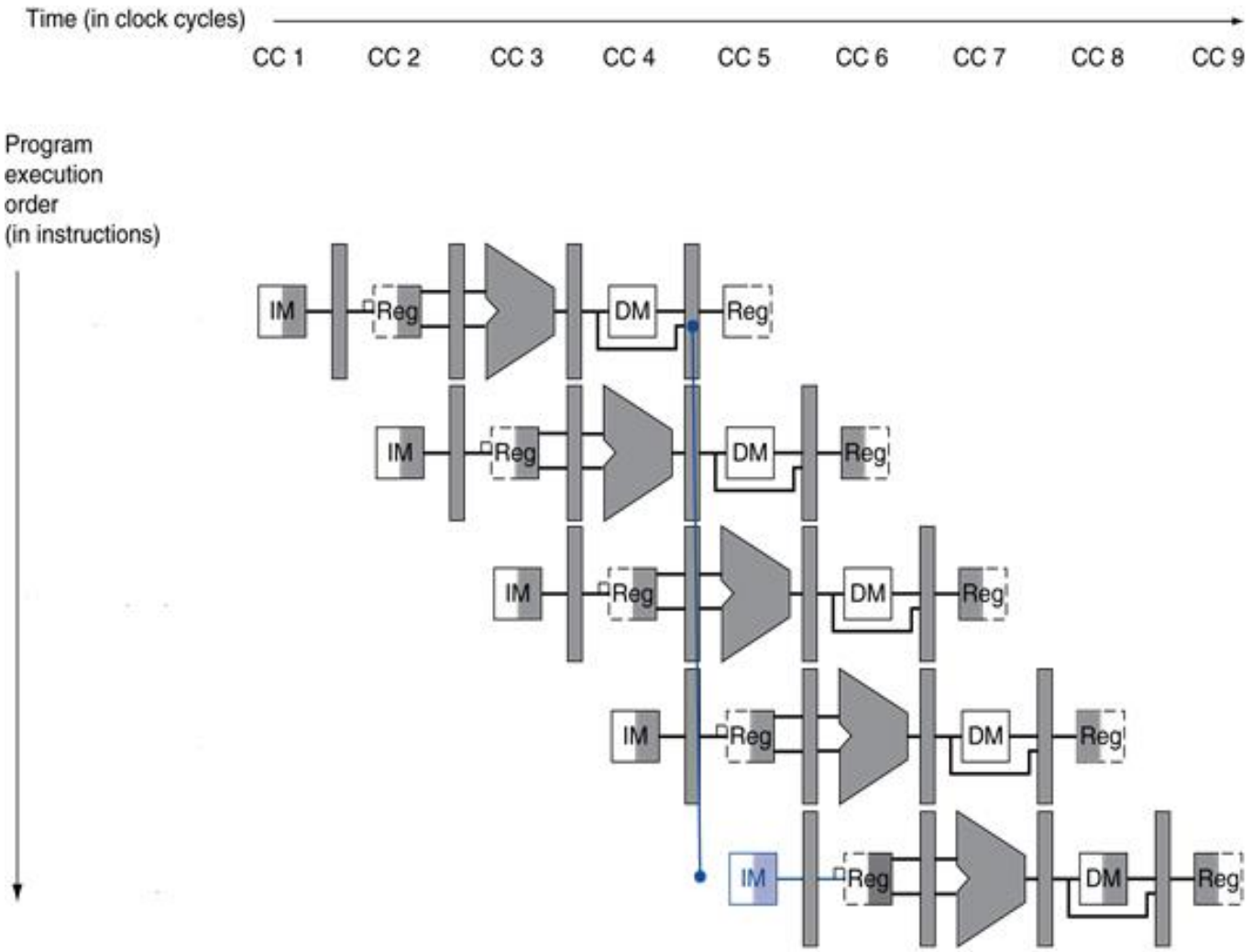

Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
IM
Reg
DM
Reg

그림 3

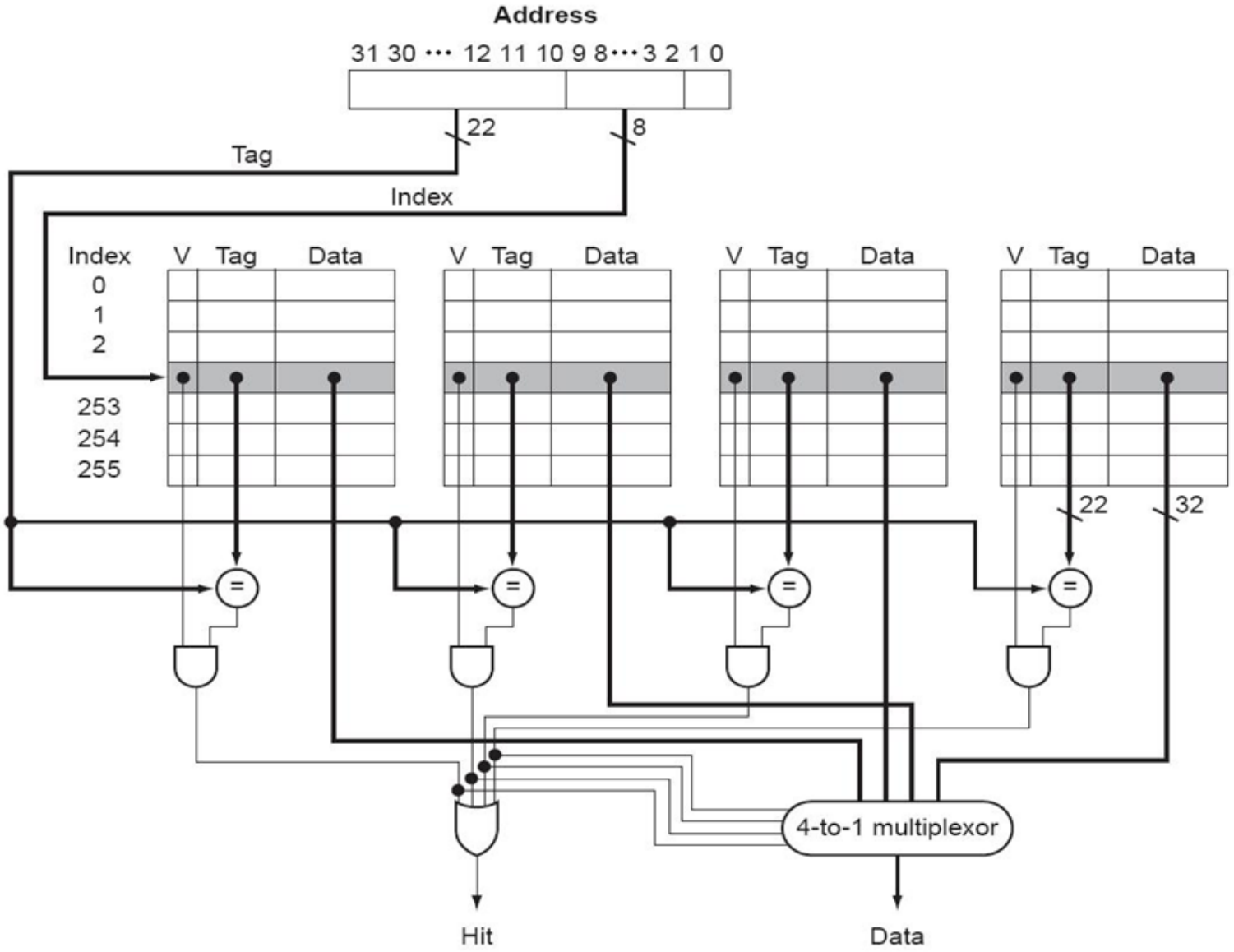

Address
31 30 ··· 12 11 10 9 8···3 2 1 0
22
8
Tag
Index
Index
V
Tag
Data
0
1
2
253
254
255
22
32
=
4-to-1 multiplexor
Hit
Data

그림 4

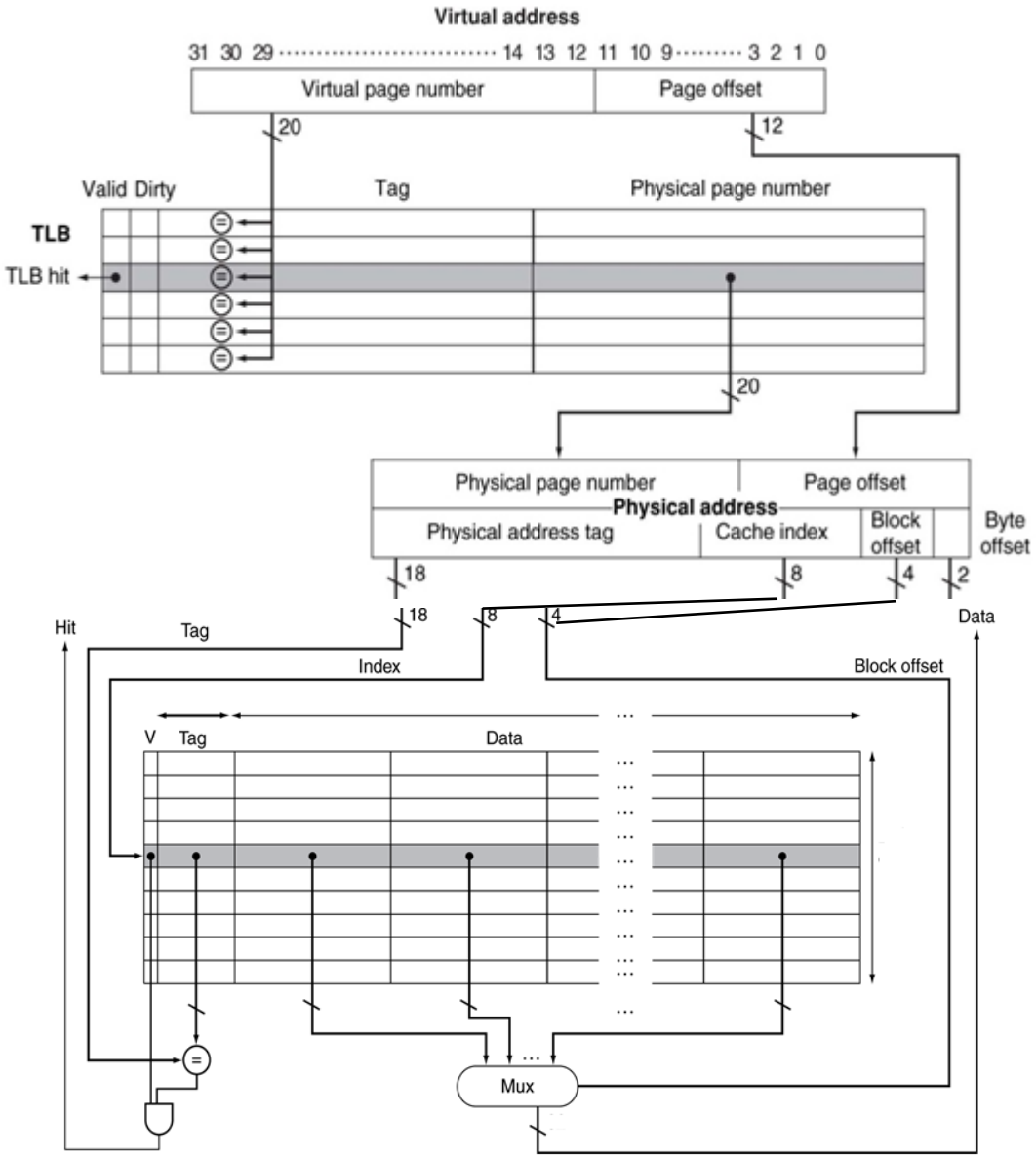

Virtual address
31 30 29 ····· 14 13 12 11 10 9 ····· 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Block offset
Byte offset
18
8
4
2
Hit
Tag
18
8
4
Data
Index
Block offset
V
Tag
Data
Mux

표 1

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

표 2

값	의미	값	의미
0	EQ (EQual)	8	HI (unsigned HIgher)
1	NE (Not Equal)	9	LS (unsigned Lowe or Same)
2	HS (unsigned Higher or Same)	10	GE (signed Greater than or Equal)
3	LO (unsigned LOwer)	11	LT (signed Less Than)
4	MI (MInus, <0)	12	GT (signed Greater Than)
5	PL (PLus, >=0)	13	LE (signed Less Than or Equal)
6	VS (oVerflow Set, overflow)	14	AL (ALways)
7	VC (oVerflow Clear, no overflow)	15	NV (reserved)

보조자료(1/3): Select Algorithm동작

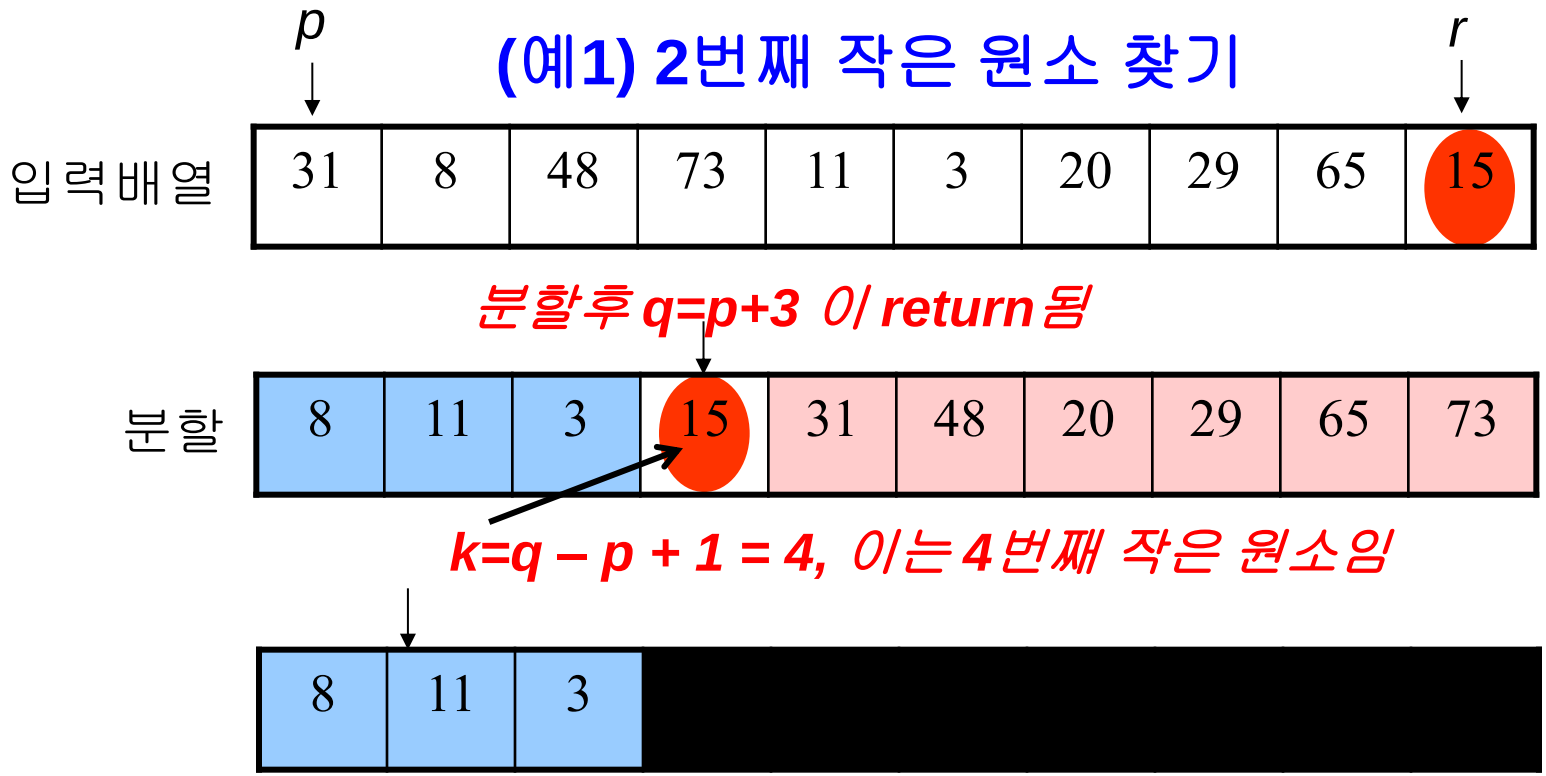
알고리즘 동작원리

1) Quicksort의 partition을 활용

2) Partition후 return 값 "q"=기준원소의 위치값을 이용하여 해당 partition을 선택한 뒤 recursive call

3) 최종적으로 원소 찾기

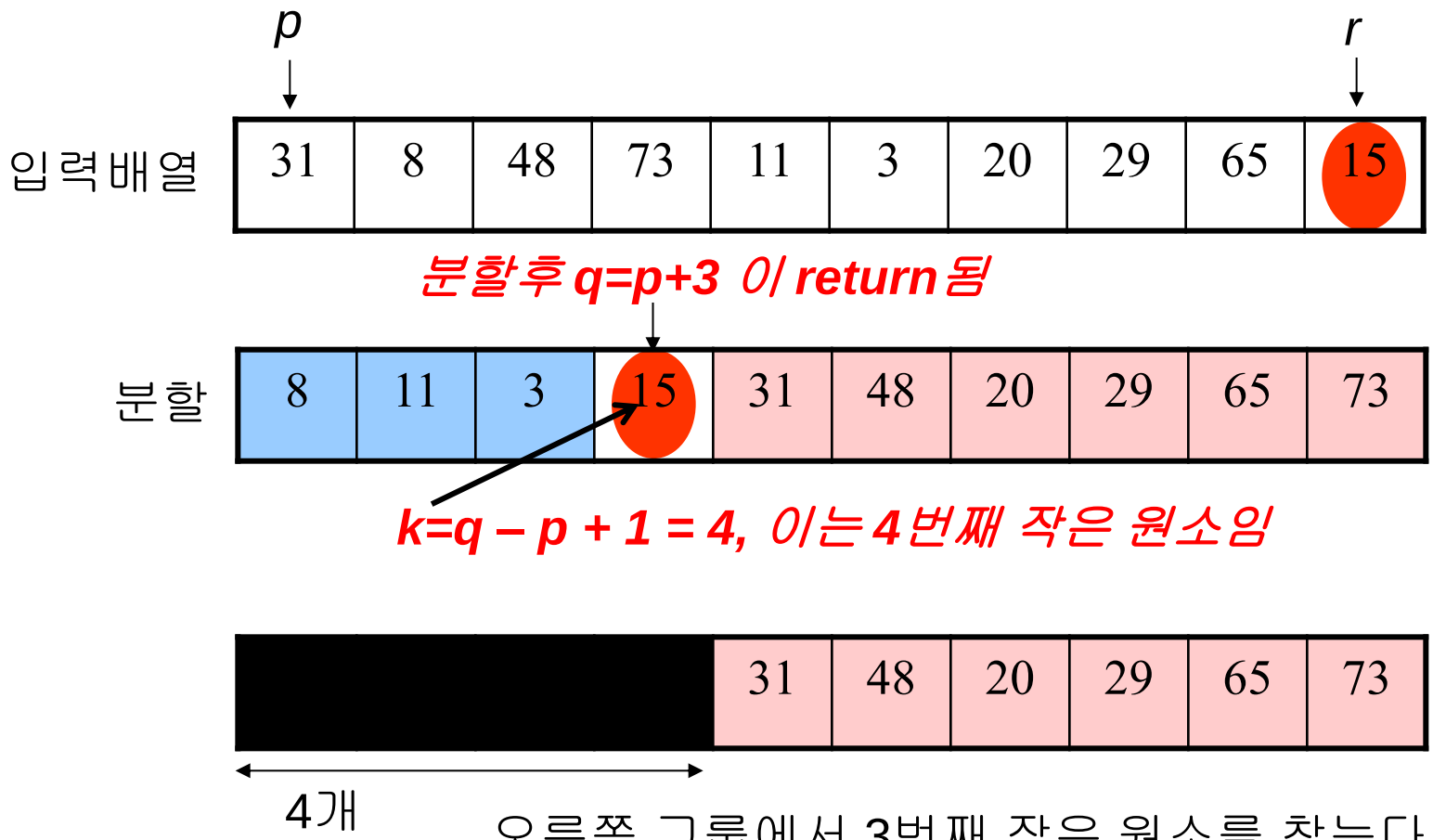
아래 예제에서는 맨 마지막 원소를 pivot로 사용



왼쪽 그룹에서 2번째 작은 원소를 찾는다

보조자료(2/3): Select Algorithm동작

(예2) 7번째 작은 원소 찾기



보조자료(3/3): quick sort

```
quicksort( array, first, last ){
partition( array, first, last );
quicksort( array, first, j-1 );
quicksort( array, j+1, last);
}
```

```
void quicksort(int array[8],int first,int
last){
   int i, j, pivot, temp;

   if(first<last){                    Partition
      pivot=first;
      i=first;
      j=last;

      while(i<j){

while(array[i]<=array[pivot]&&i<last)
            i++;
         while(array[j]>array[pivot])
            j--;
         if(i<j){
            temp=array[i];
            array[i]=array[j];
            array[j]=temp;
         }
      }

      temp=array[pivot];
      array[pivot]=array[j];
      array[j]=temp;
      quicksort(array,first,j-1);
      quicksort(array,j+1,last);

   }
}
```