COL216
Computer Architecture

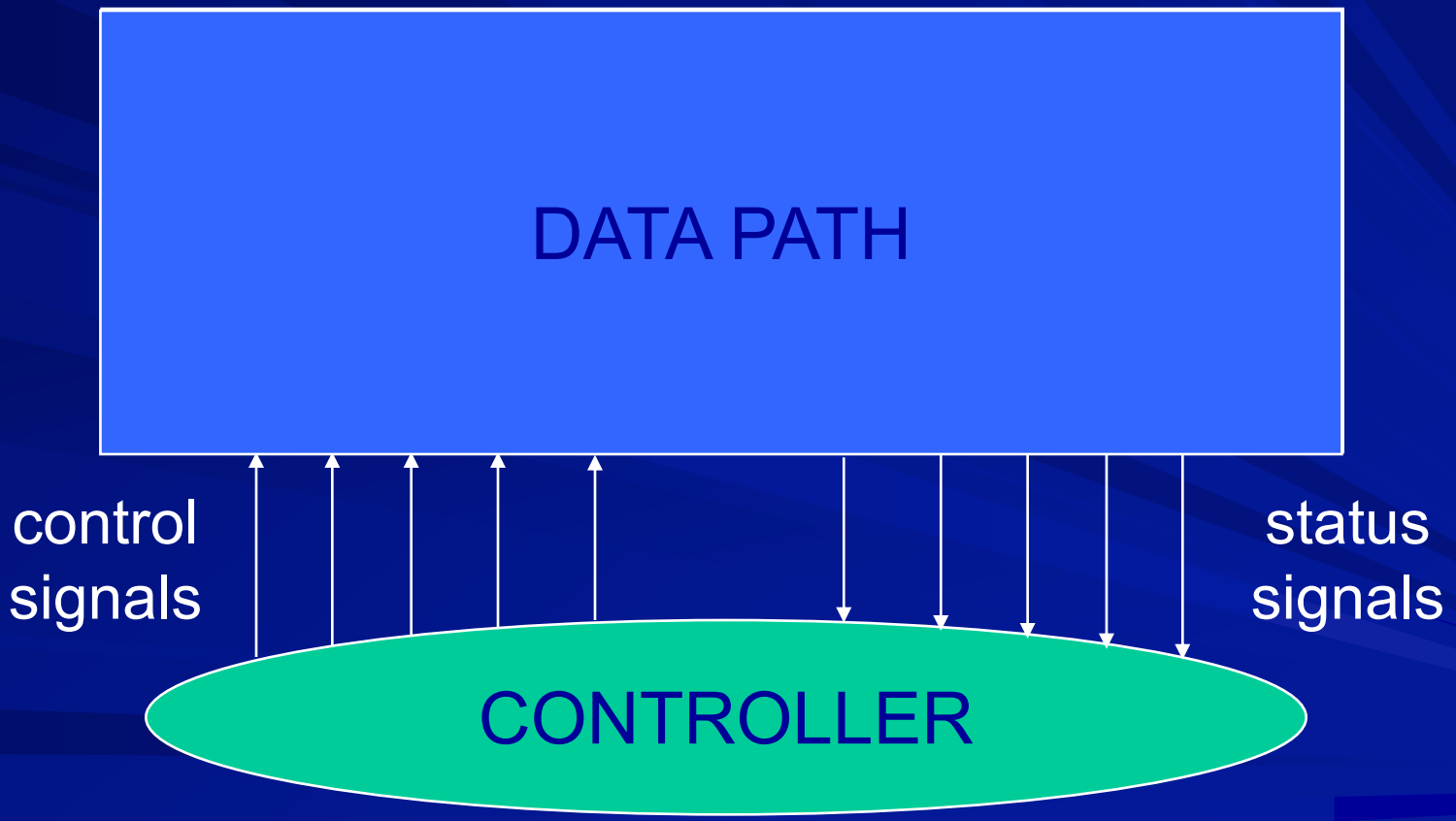
Designing a processor:
Datapath building blocks
31st January, 2022

Overview

- Given an ISA, how to design a Micro architecture
- There are numerous possibilities
- Different combinations of performance – cost – power consumption are possible
- CAD tools help in analyzing the trade offs between these parameters
- CAD tools are required for physical implementation

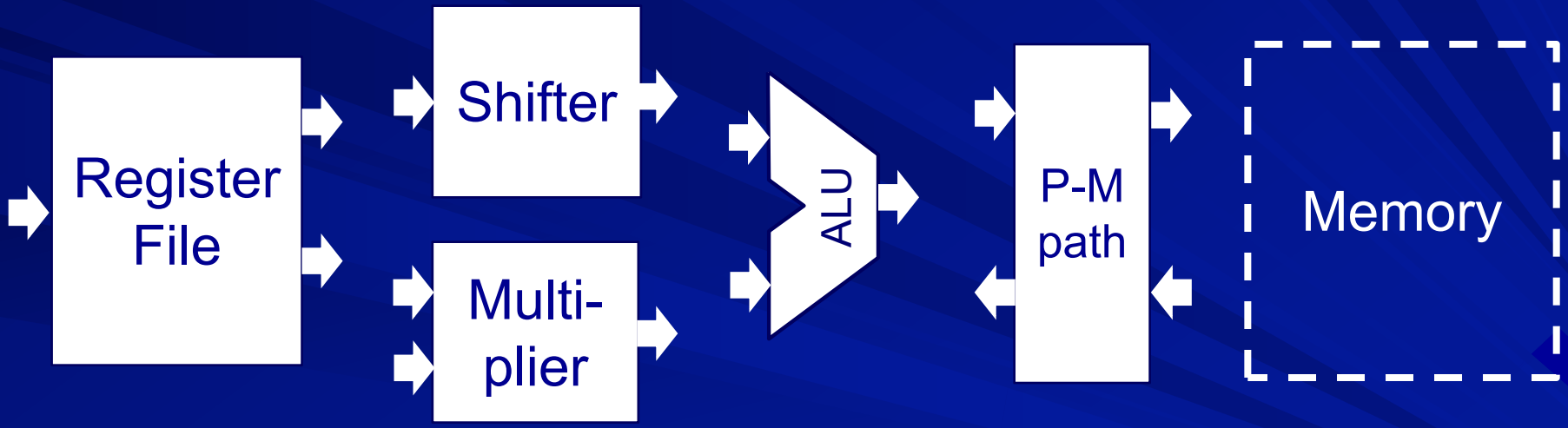
CPU datapath + controller

Focus of this lecture: Major building blocks for datapath



Datapath major blocks

Some additional blocks are required to glue these together



Later we will see how instruction fetching, instruction decode, operand access and state updation are done

ARM instruction subset

adc sbc	add sub	rsb	rsc
and	bic	eor	orr
cmn	cmp	teq	tst
mov	mvn		
mla	mul		
ldr	str		
b	bl		
swi			

Instructions excluded - 1

- cdp : co-processor data processing
- mcr : move from CPU register to co-processor register
- mrc : move from co-processor register to CPU register
- ldc : load co-processor register from memory
- stc : store co-processor register to memory

Instructions excluded - 2

- bx : branch and exchange
  - switch between ARM and Thumb modes
- ldm : load multiple registers
- stm : store multiple registers
- swp : swap register - memory (atomically)

Instruction classes

- Data processing (DP)
  - arithmetic
  - logical
  - test
  - move
- Branch
- Multiply
- Data transfer (DT)

ALU operation for each class

- Data processing (DP)
  - arithmetic
  - logical
  - test
  - move
- Branch
- Multiply
- Data transfer (DT)

- Perform specified
  arithmetic or
  logical or relational operation or no operation
- Target address
- Multiply {and add}
- Memory address

ALU operations for DP instructions

Instr	ins [24-21]	Operation
and	0 0 0 0	Op1 AND Op2
eor	0 0 0 1	Op1 EOR Op2
sub	0 0 1 0	Op1 + NOT Op2 + 1
rsb	0 0 1 1	NOT Op1 + Op2 + 1
add	0 1 0 0	Op1 + Op2
adc	0 1 0 1	Op1 + Op2 + C
sbc	0 1 1 0	Op1 + NOT Op2 + C
rsc	0 1 1 1	NOT Op1 + Op2 + C
tst	1 0 0 0	Op1 AND Op2
teq	1 0 0 1	Op1 EOR Op2
cmp	1 0 1 0	Op1 + NOT Op2 + 1
cmn	1 0 1 1	Op1 + Op2
orr	1 1 0 0	Op1 OR Op2
mov	1 1 0 1	Op2
bic	1 1 1 0	Op1 AND NOT Op2
mvn	1 1 1 1	NOT Op2

ALU operations for other instructions

Instr	ins [23] : U	Operation
ldr	1	Op1 + Op2
ldr	0	Op1 + NOT Op2 + 1
str	1	Op1 + Op2
str	0	Op1 + NOT Op2 + 1
b		Op1 + Op2 + k
bl		Op1 + Op2 + k
mul		Op1 * Op2
mla		Op1 * Op2 + Op3

Instruction variations / suffixes

- Suffixes
  - Predication
  - Setting flags
  - Word / half word / byte transfer
- Operand variations
  - Shifting / rotation
  - Pre / post increment / decrement
  - Auto increment / decrement

Instructions with Suffixes

Arithmetic: <add|sub|rsb|adc|sbc|rsc> {*cond*} {s}

Logical: <and | orr | eor | bic> {*cond*} {s}

Test: <cmp | cmn | teq | tst> {*cond*}

Move: <mov | mvn> {*cond*} {s}

Branch: <b | bl> {*cond*}

Multiply: <mul | mla> {*cond*} {s}

Load/store: <ldr | str> {*cond*} {b | h | sb | sh }

Shift/rotate in DP instructions

- 12 bit operand2 in DP instructions

4 2 1 4 1 4 4 12

– 8 bit unsigned number,
  4 bit rotate spec, or
– 4 bit register number,

  8 bit shift specification
  shift type: LSL, LSR, ASR, ROR
  shift amount: 5 bit constant or 4 bit register no.

Shift/rotate in DT instructions

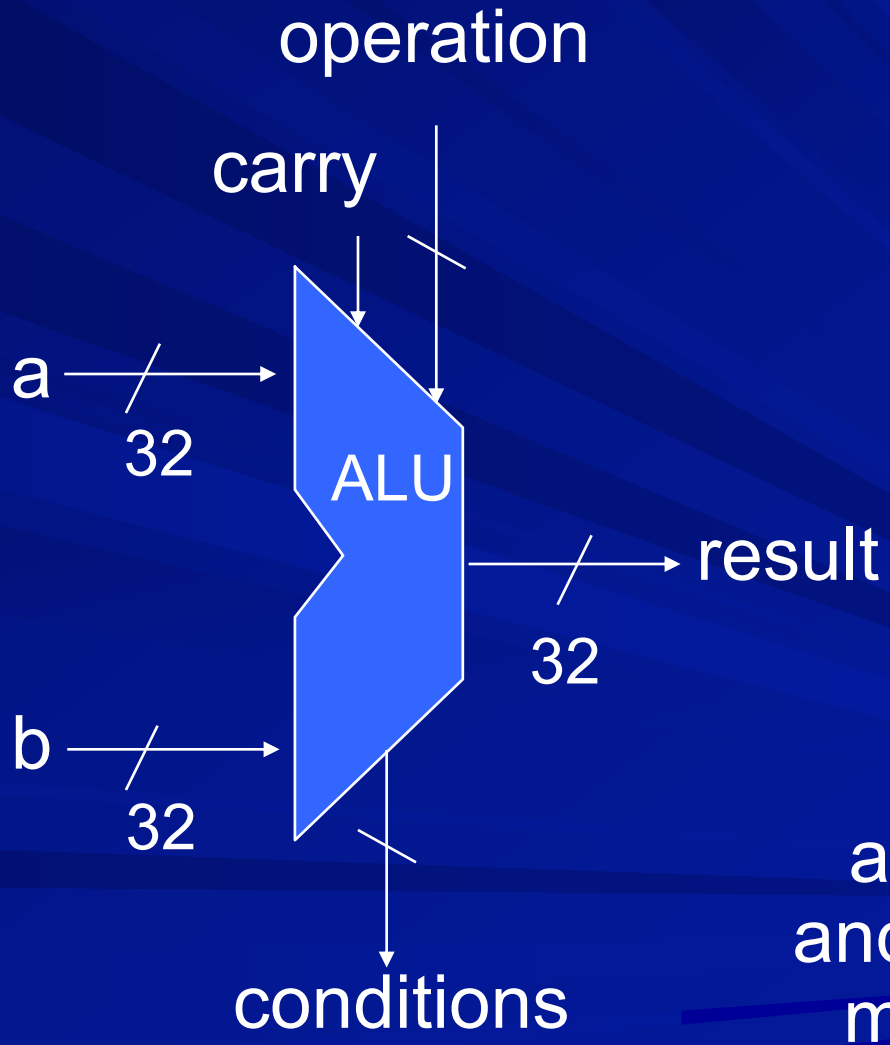
- 12 bit offset field in DT instructions

– 12 bit unsigned constant

or

– 4 bit register number, 8 bit shift specification (same format as DP instructions)

Arithmetic and Logical operations



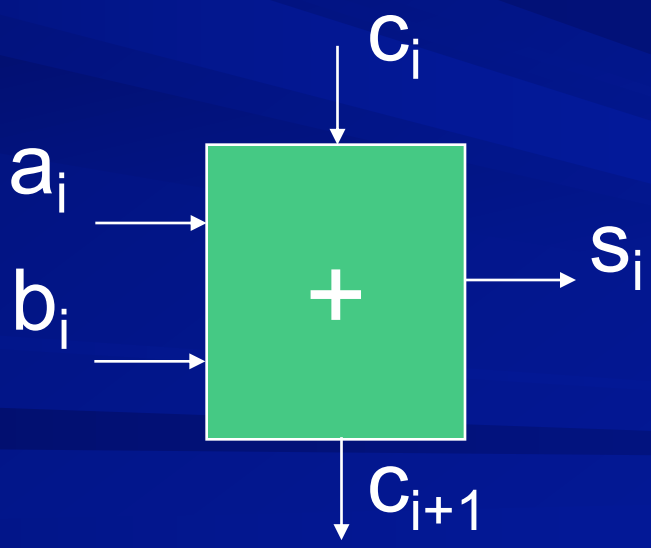
sources of operands

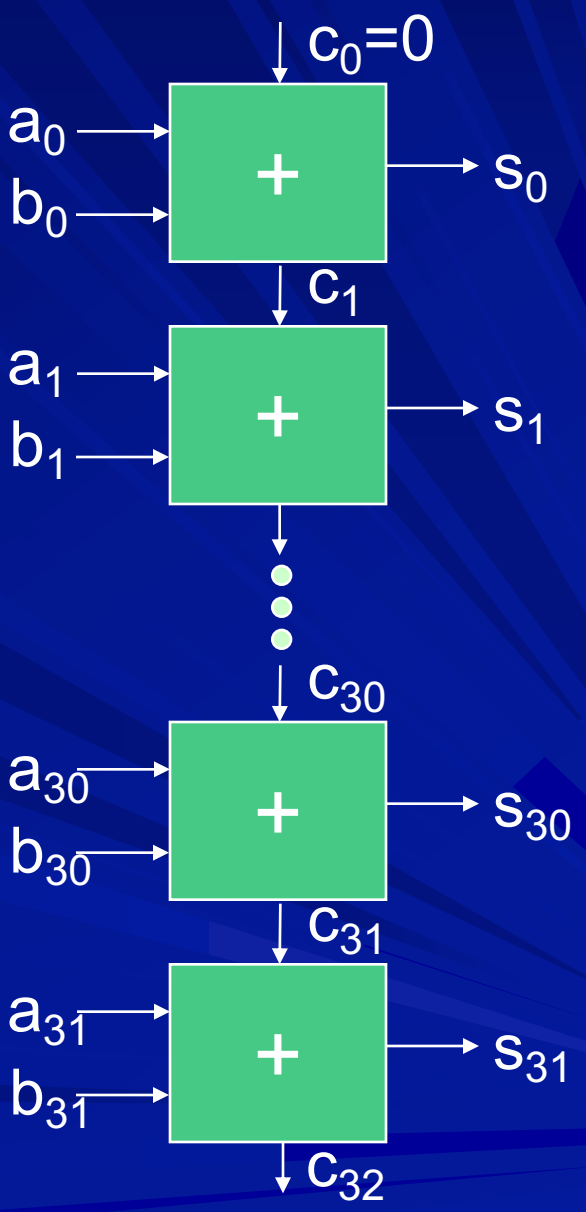
destination of result

add, sub, and, orr, eor, mov, mvn

Adder circuit

- Perform addition with carry propagation or carry look ahead

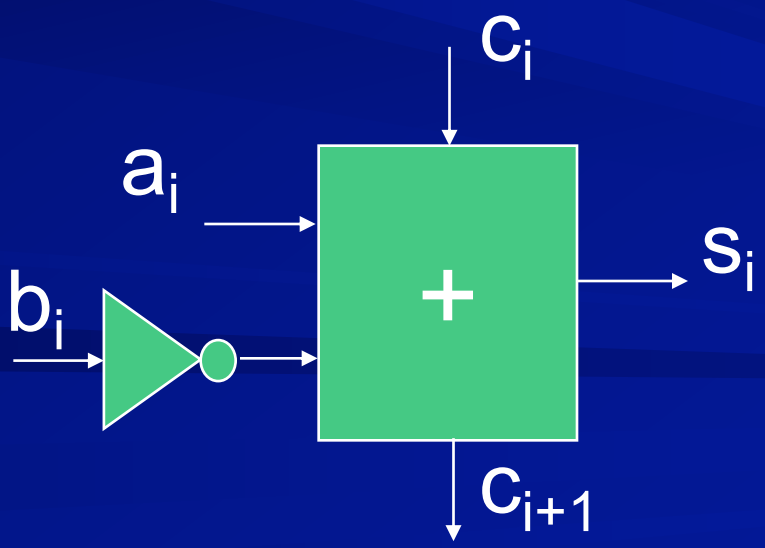


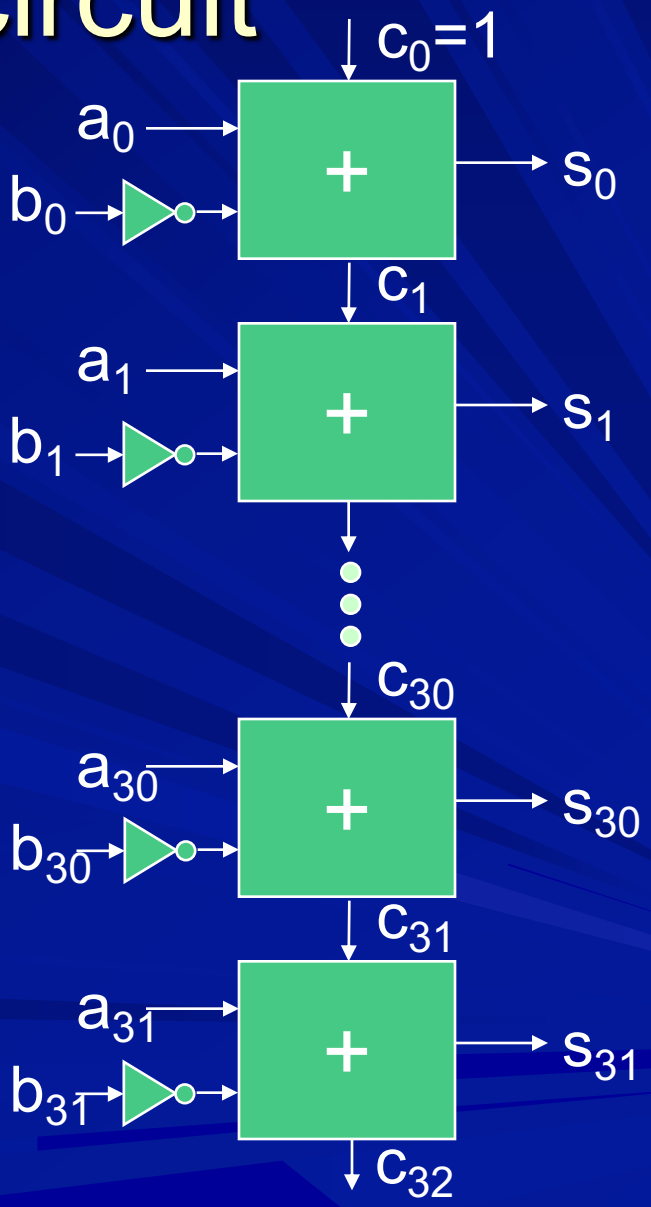


Subtraction circuit

- Use the formula

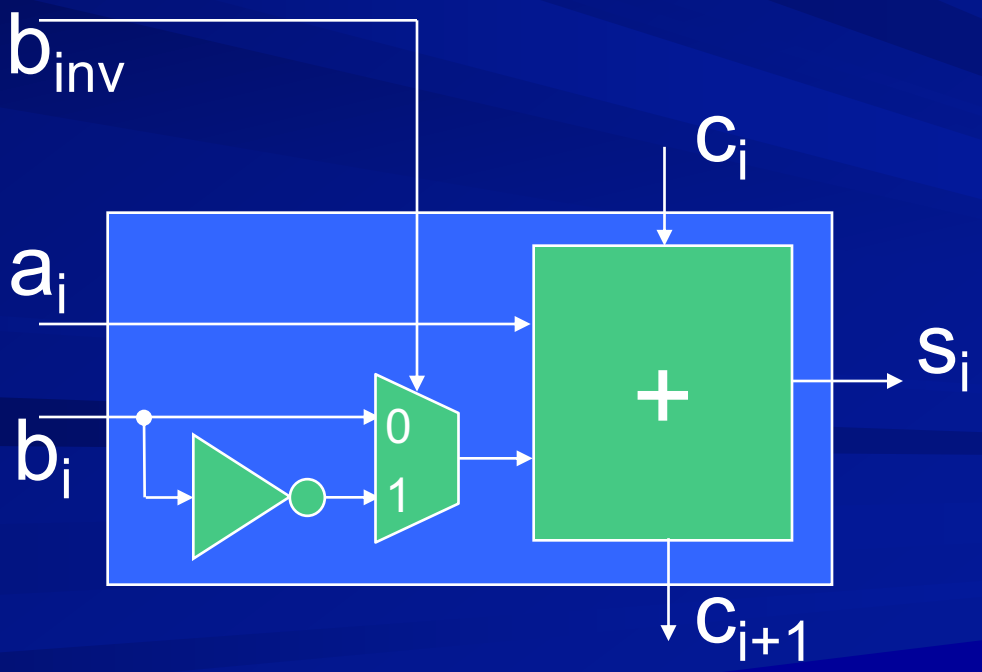
  X-Y = X+Y' +1

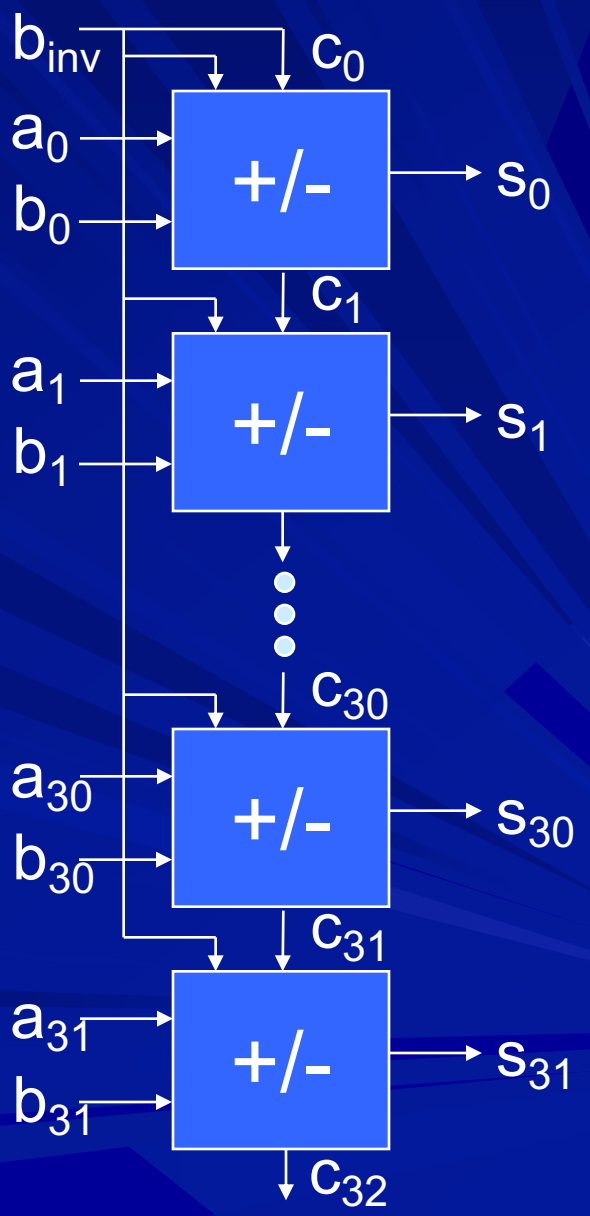




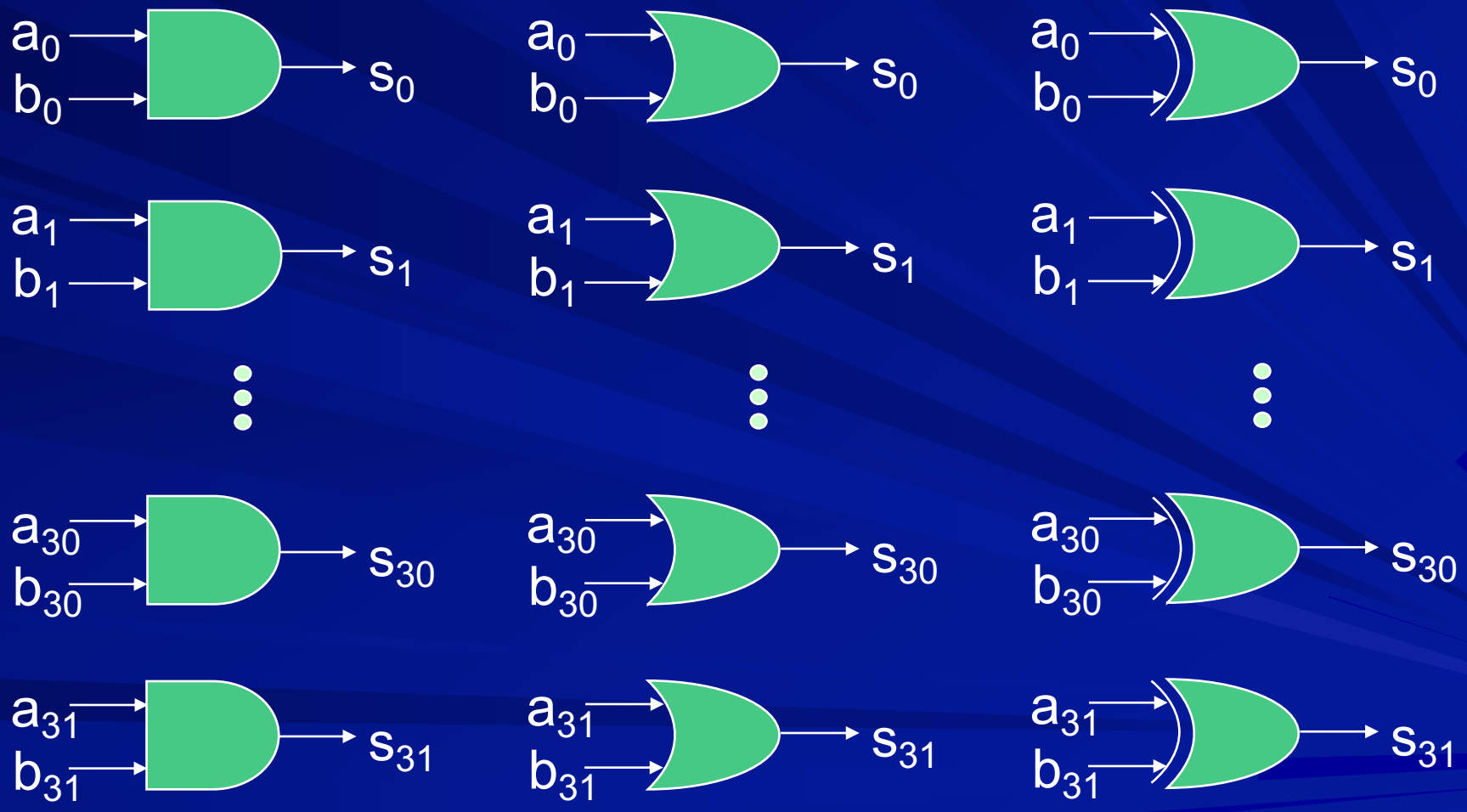
Combining addition and subtraction

- Use a multiplexer circuit

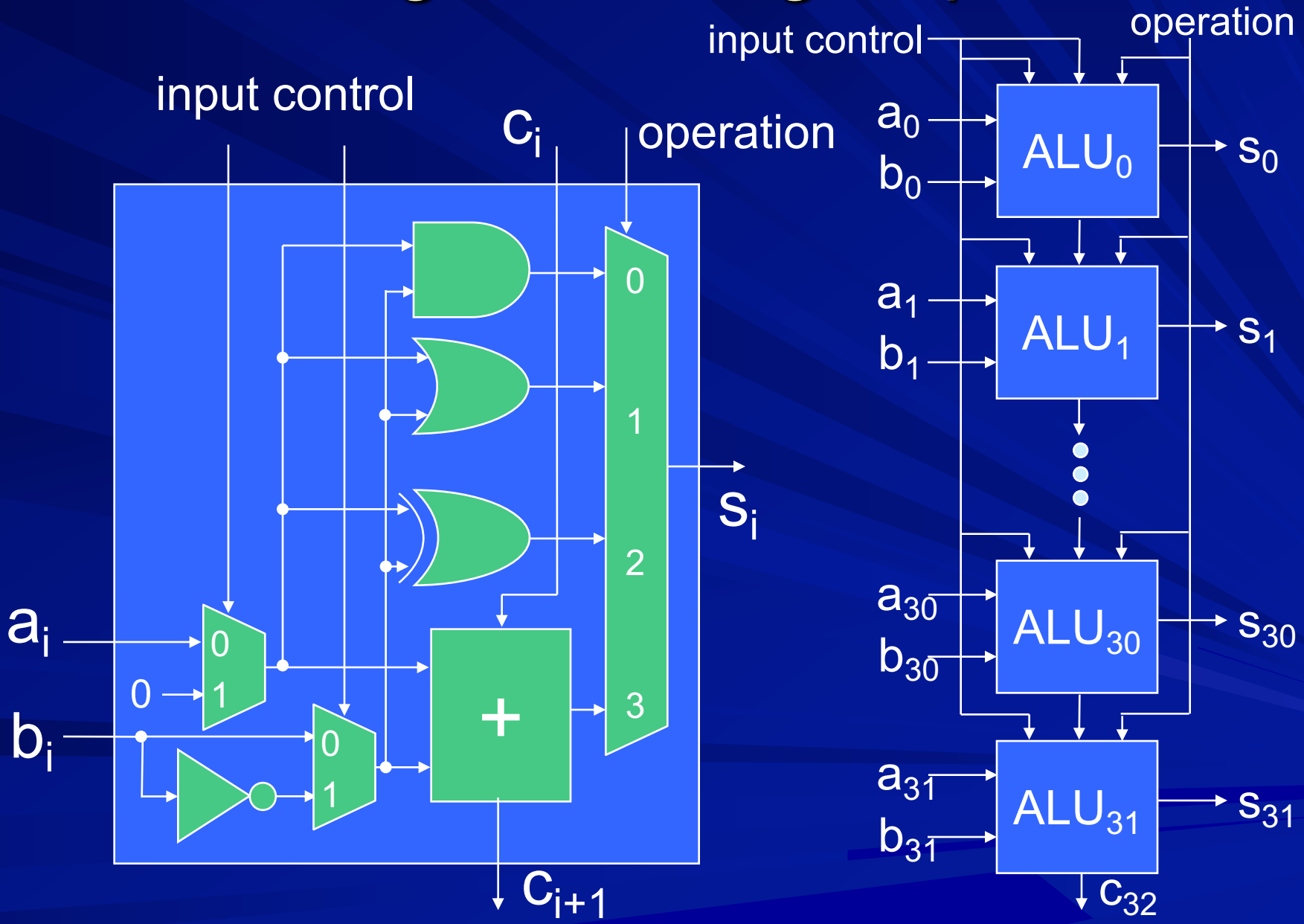




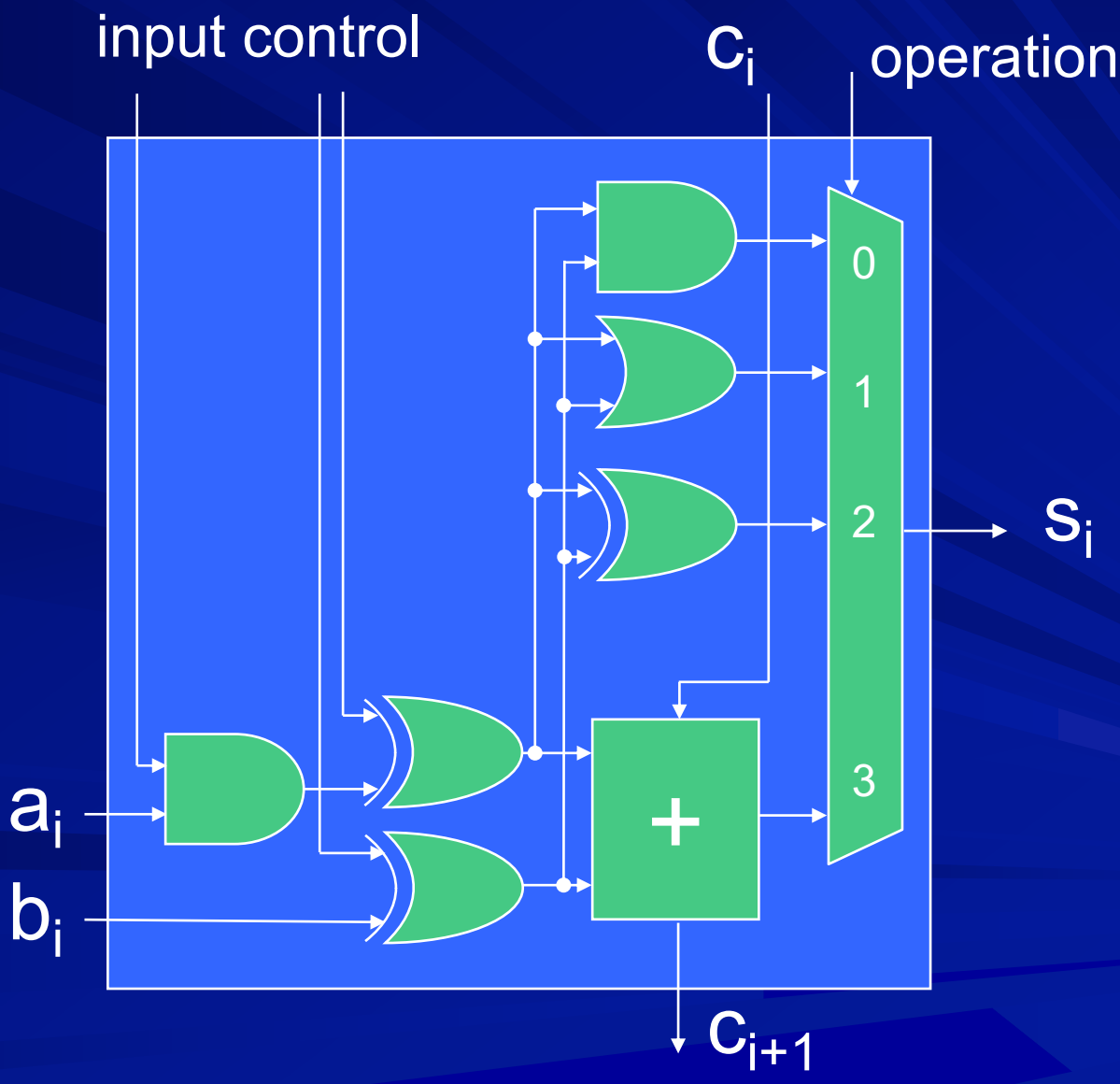
Circuit for and, or, xor



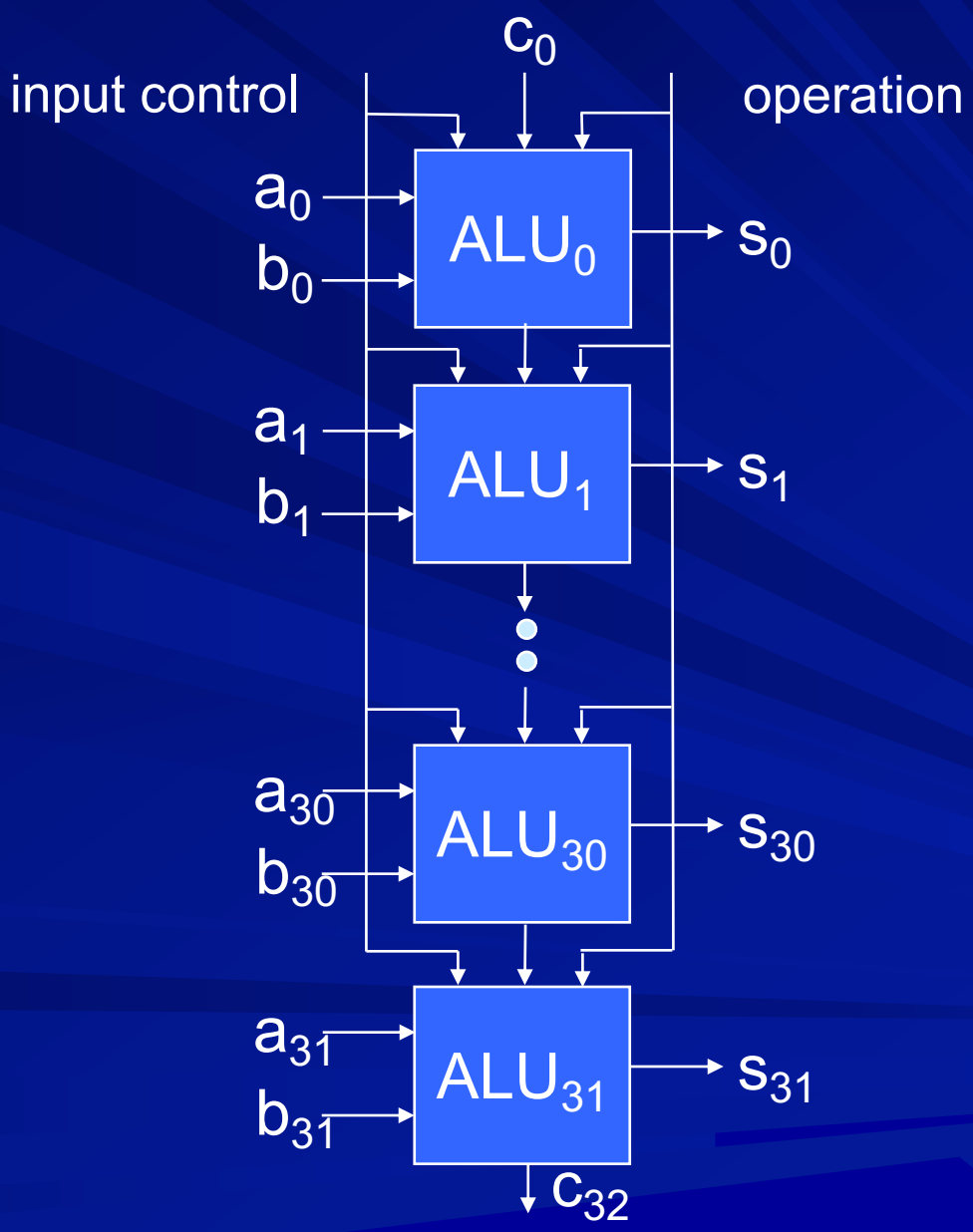
Combining arith & logic operations



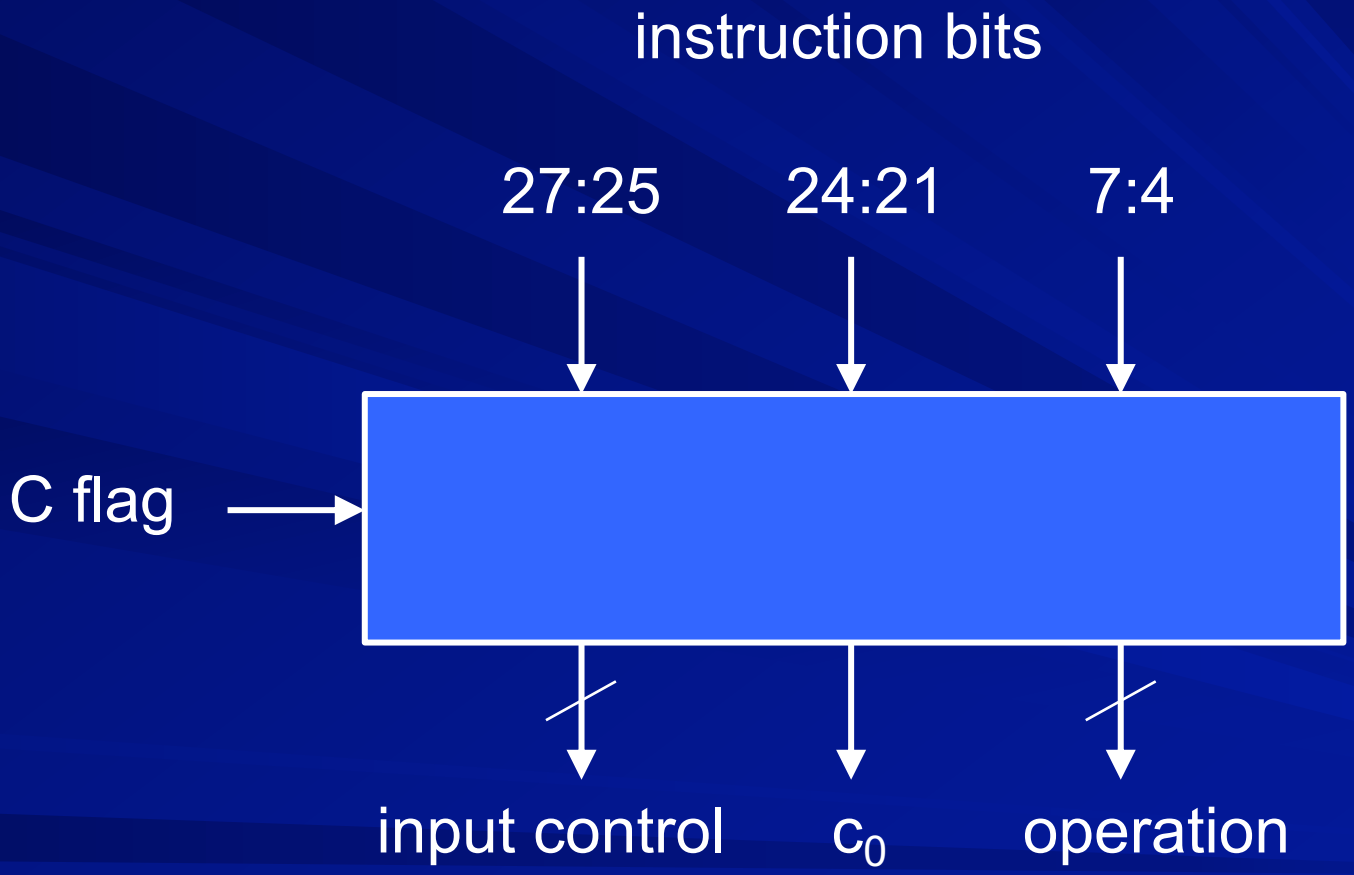
ALU Cell with reverse subtract



ALU – 32 bits

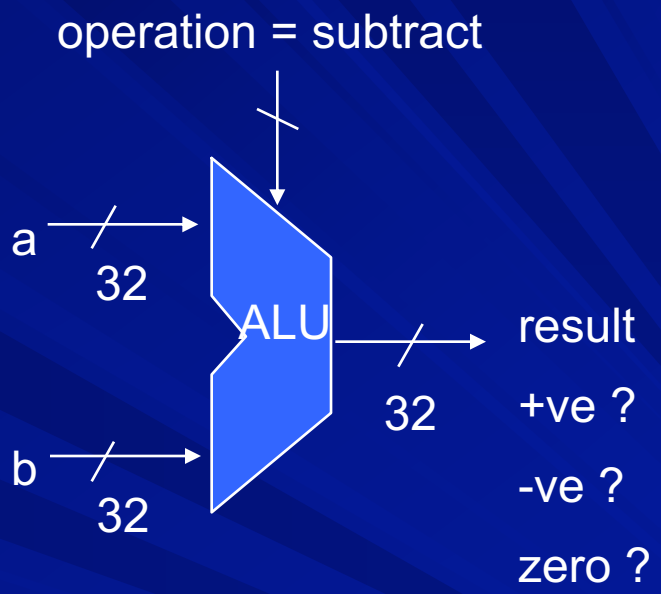

c_0
input control
operation
a_0
b_0
ALU_0
s_0
a_1
b_1
ALU_1
s_1
a_{30}
b_{30}
ALU_{30}
s_{30}
a_{31}
b_{31}
ALU_{31}
s_{31}
c_{32}

ALU control inputs

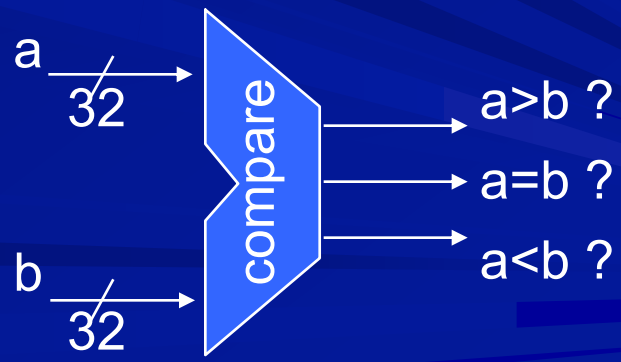


Comparing two integers

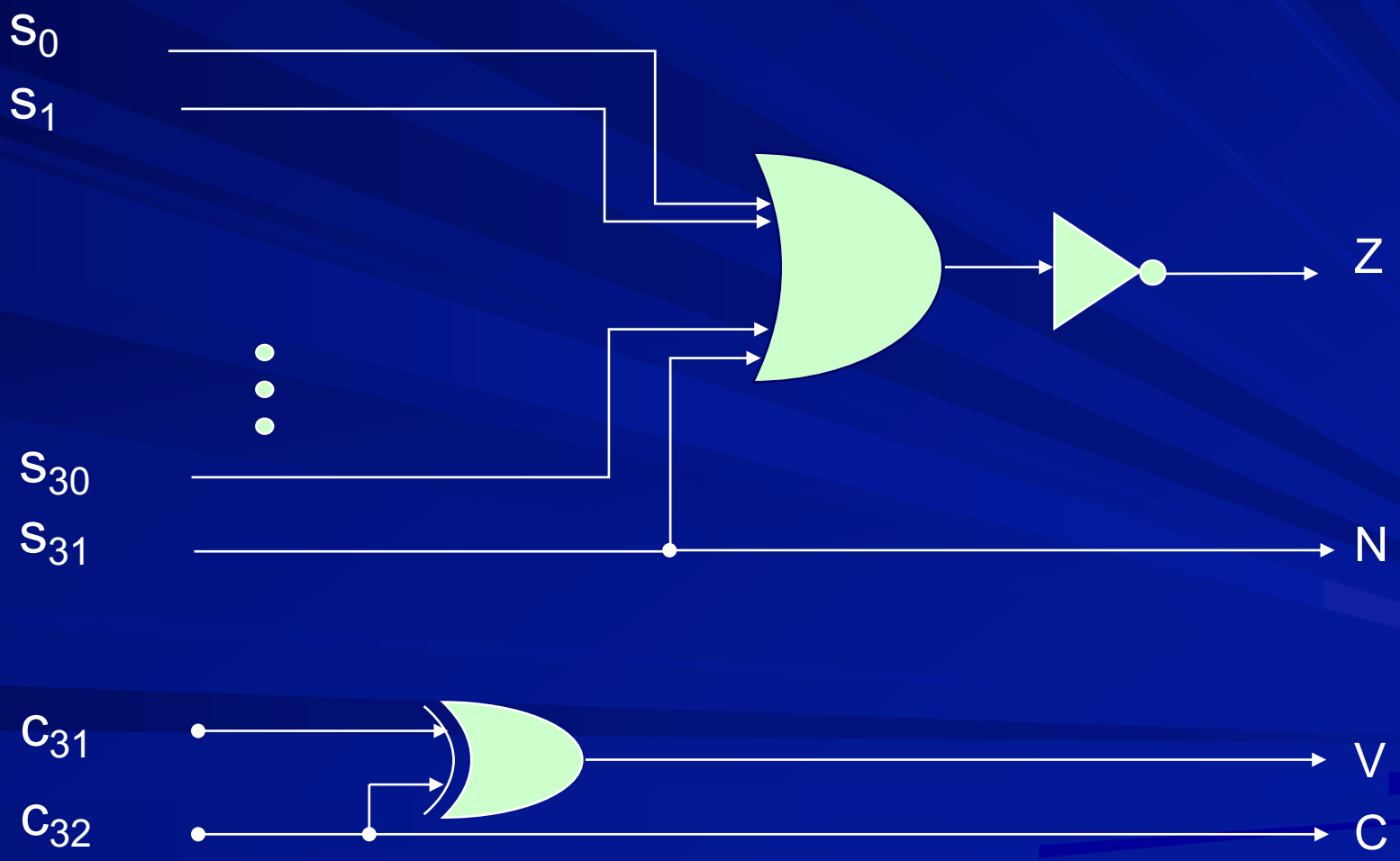
- Subtract and check the result



- Compare directly



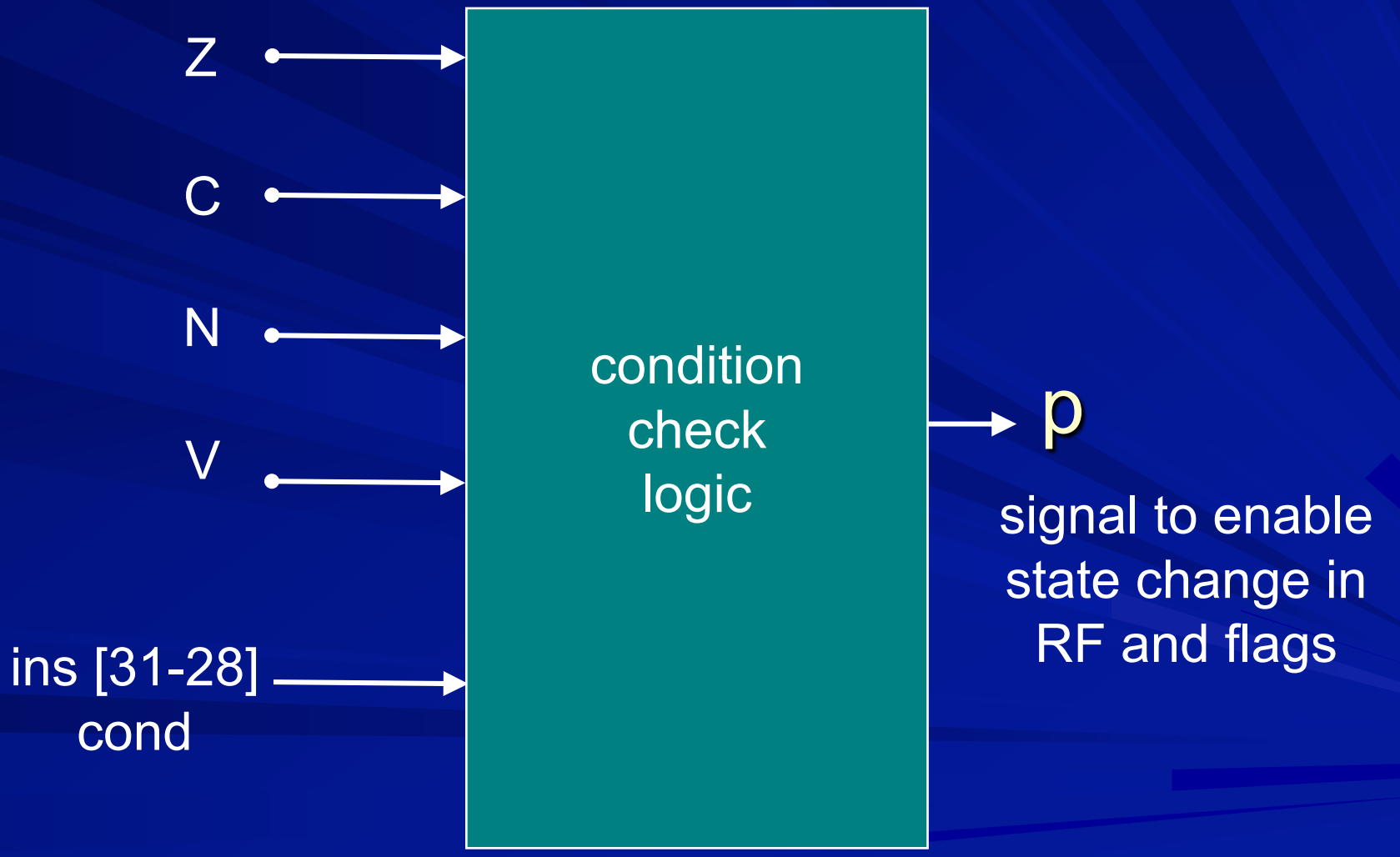
Setting flags



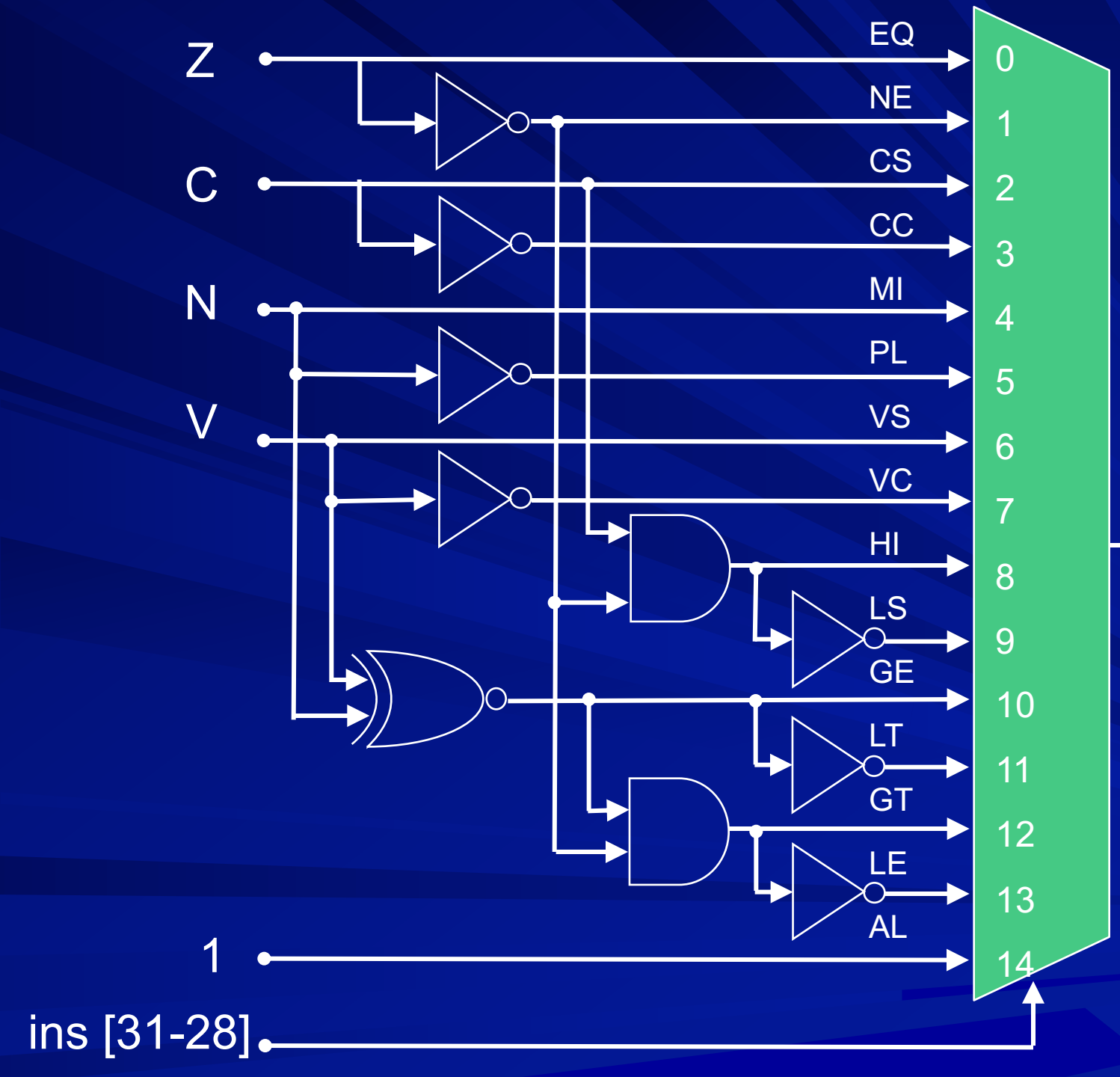
Predication

cond	ins [31-28]	Flags
EQ	0 0 0 0	Z set
NE	0 0 0 1	Z clear
CS | HS	0 0 1 0	C set | higher or same
CC | LO	0 0 1 1	C clear | lower
MI	0 1 0 0	N set
PL	0 1 0 1	N clear
VS	0 1 1 0	V set
VC	0 1 1 1	V clear
HI	1 0 0 0	C set and Z clear
LS	1 0 0 1	C clear or Z set
GE	1 0 1 0	N = V
LT	1 0 1 1	N ≠ V
GT	1 1 0 0	Z clear and (N = V)
LE	1 1 0 1	Z set or (N ≠ V)
AL	1 1 1 0	ignored

Predication



Predi-cation



Instructions doable by earlier ALU

- add, sub
- and, eor, orr, bic
- cmp, cmn
- tst, teq
- mov, mvn

Additional DP instructions to be done

- adc, sbc
- rsb, rsc

For reverse subtraction, we can either include multiplexers to switch between Op1 and Op2, or include option to invert Op1.

The initial carry can be constant ‘0’, constant ‘1’ or C Flag.

Op2 variants

- Rm

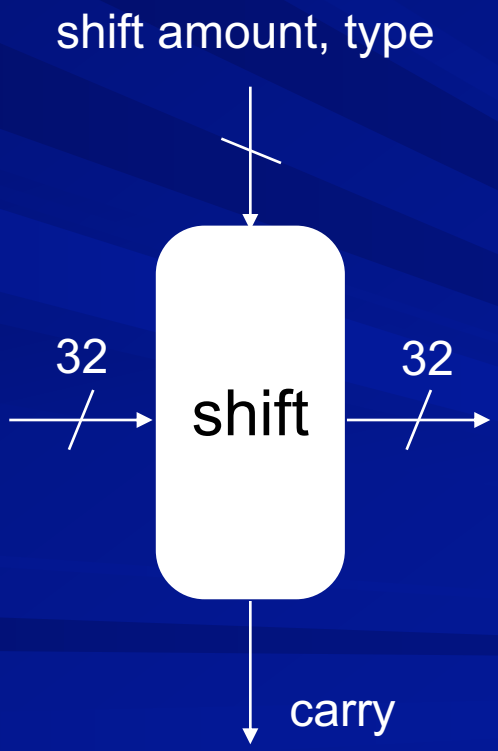
⇒Rm, <Shift type>, Rs

⇒Rm, <Shift type>, #constant

- #constant

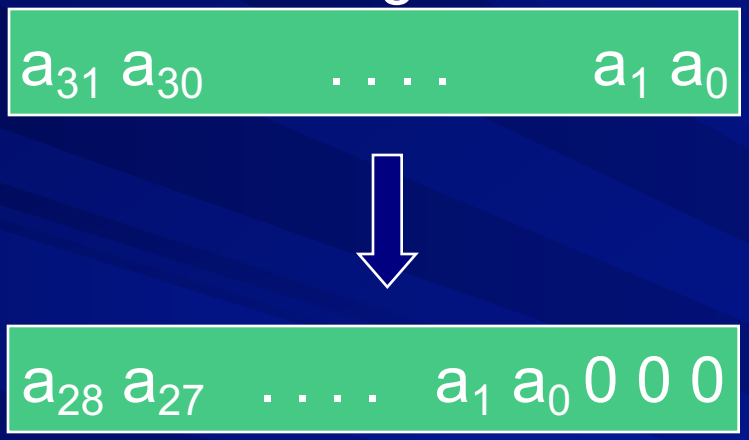
⇒#constant, ROR, constant

Shifting and rotation



Shift operations

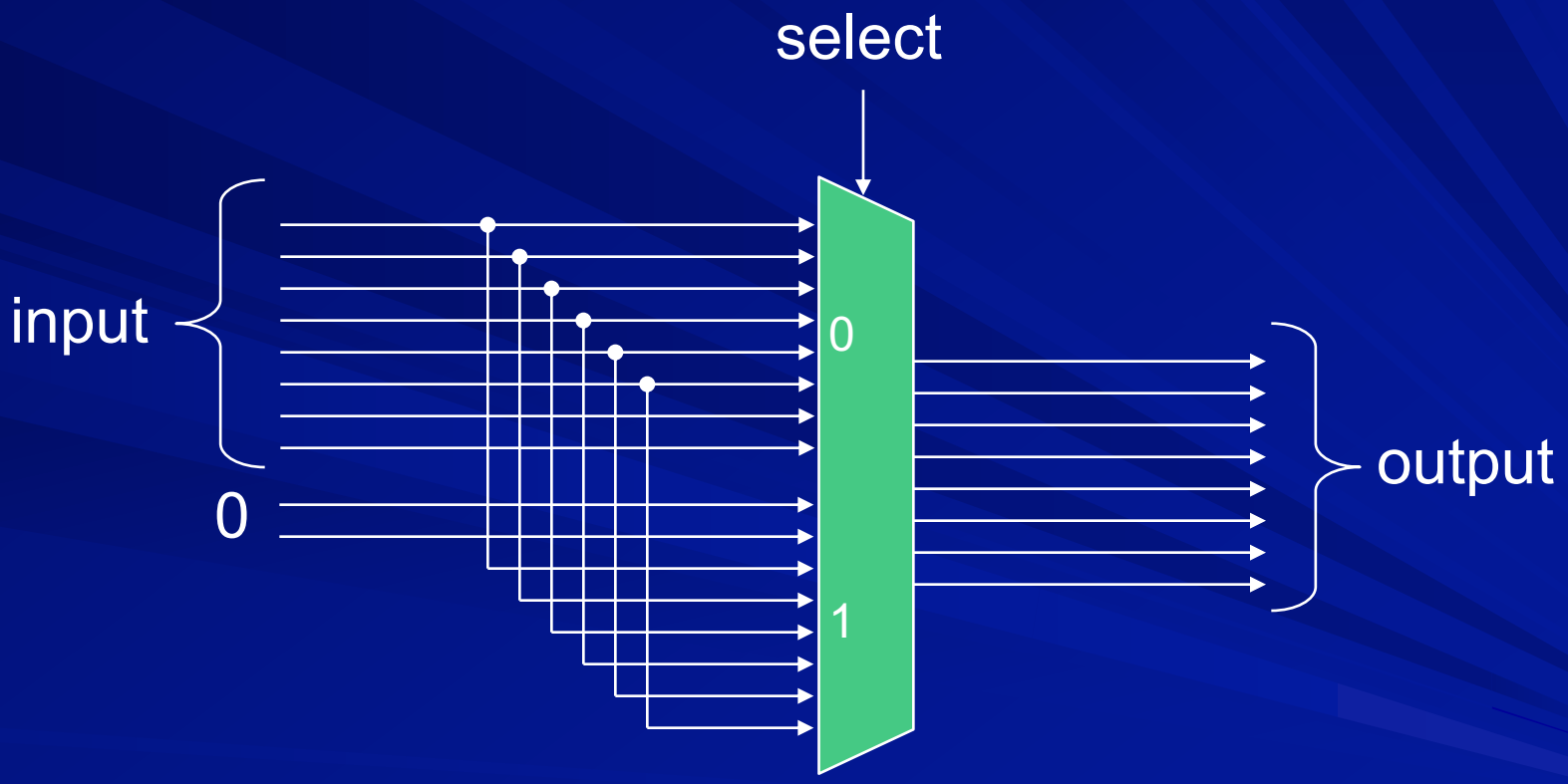
shift left logical 3 bits



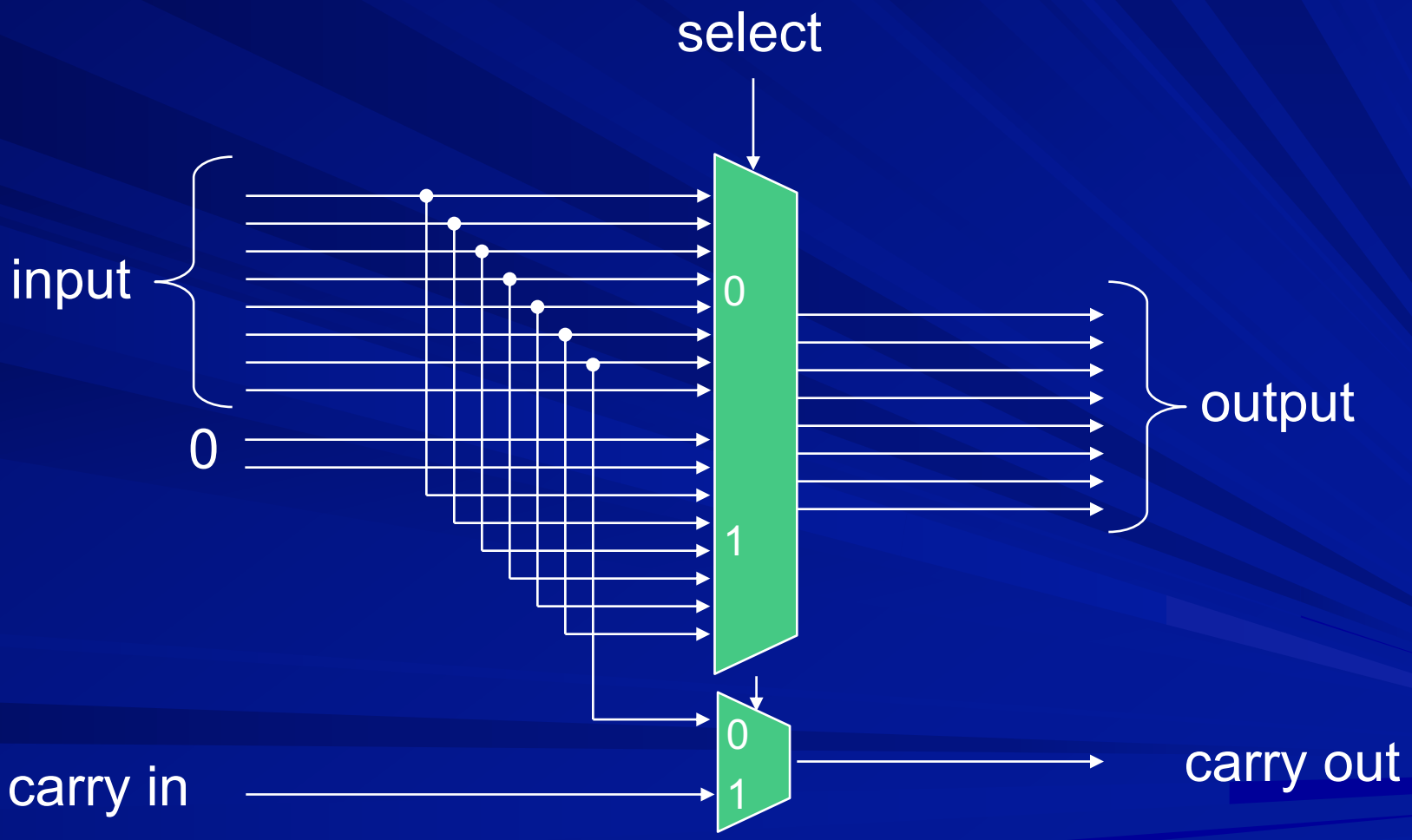
shift right logical 3 bits

shift right arithmetic 2 bits

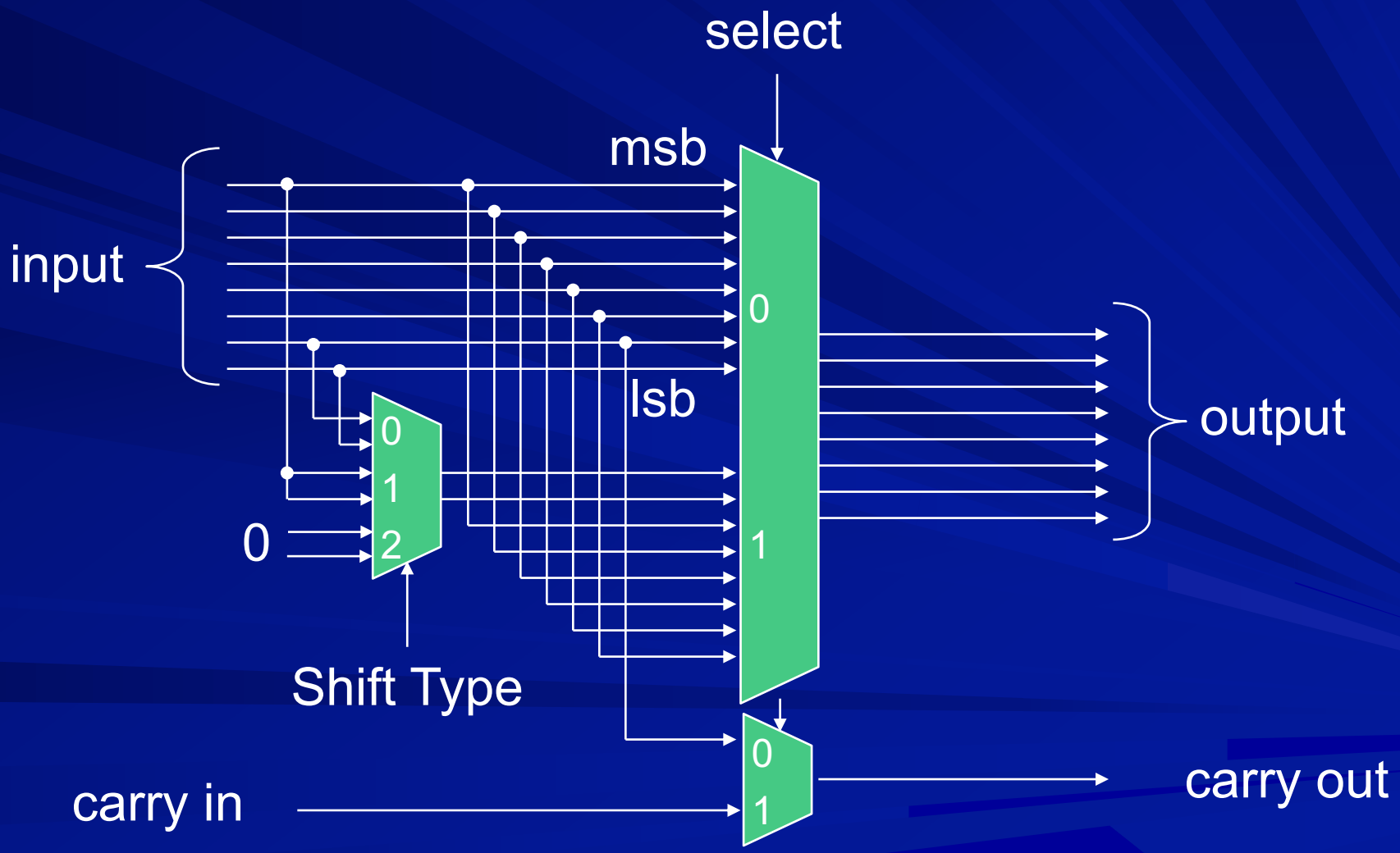
Circuit for shifting by 2 bits



Circuit for shifting by 2 bits

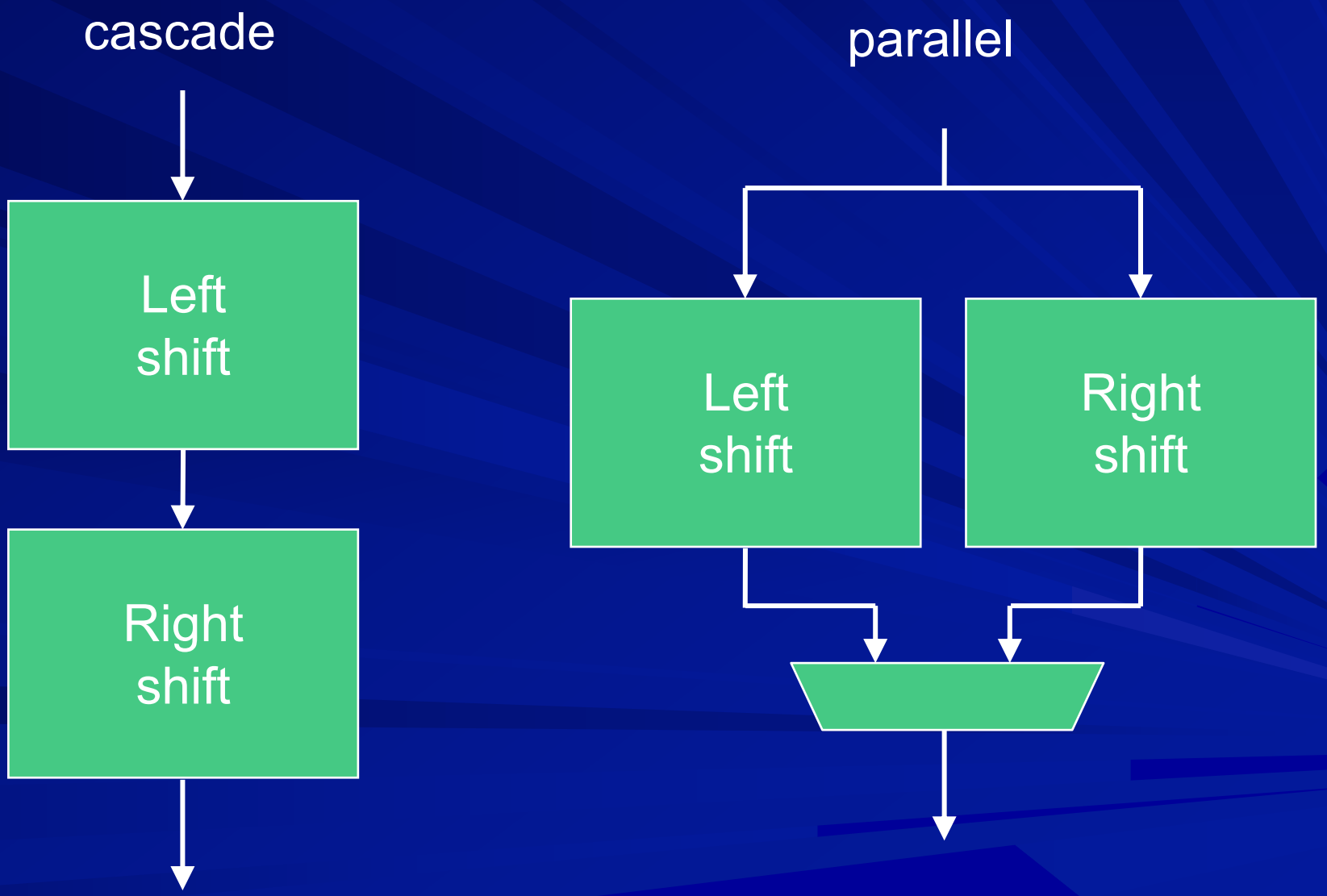


Combining ROR, ASR, LSR

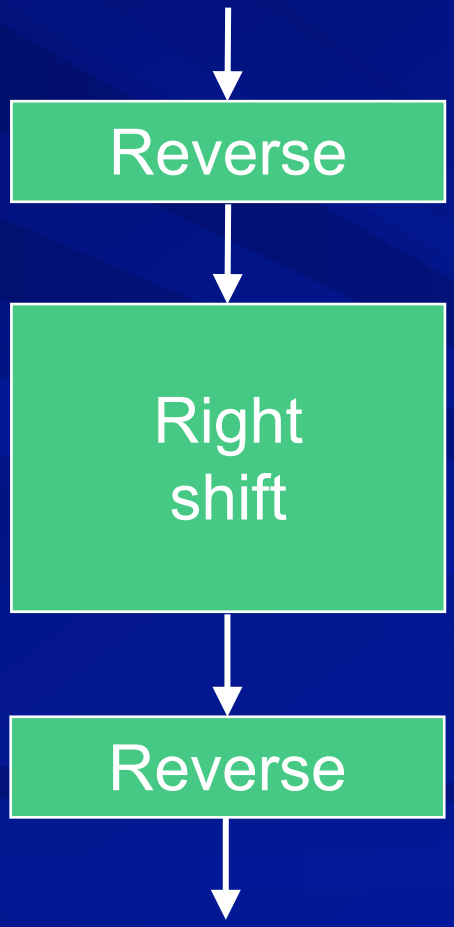


Circuit for shifting by 0 to 31 bits

Combining Left and Right Shift



Combining Left and Right Shift



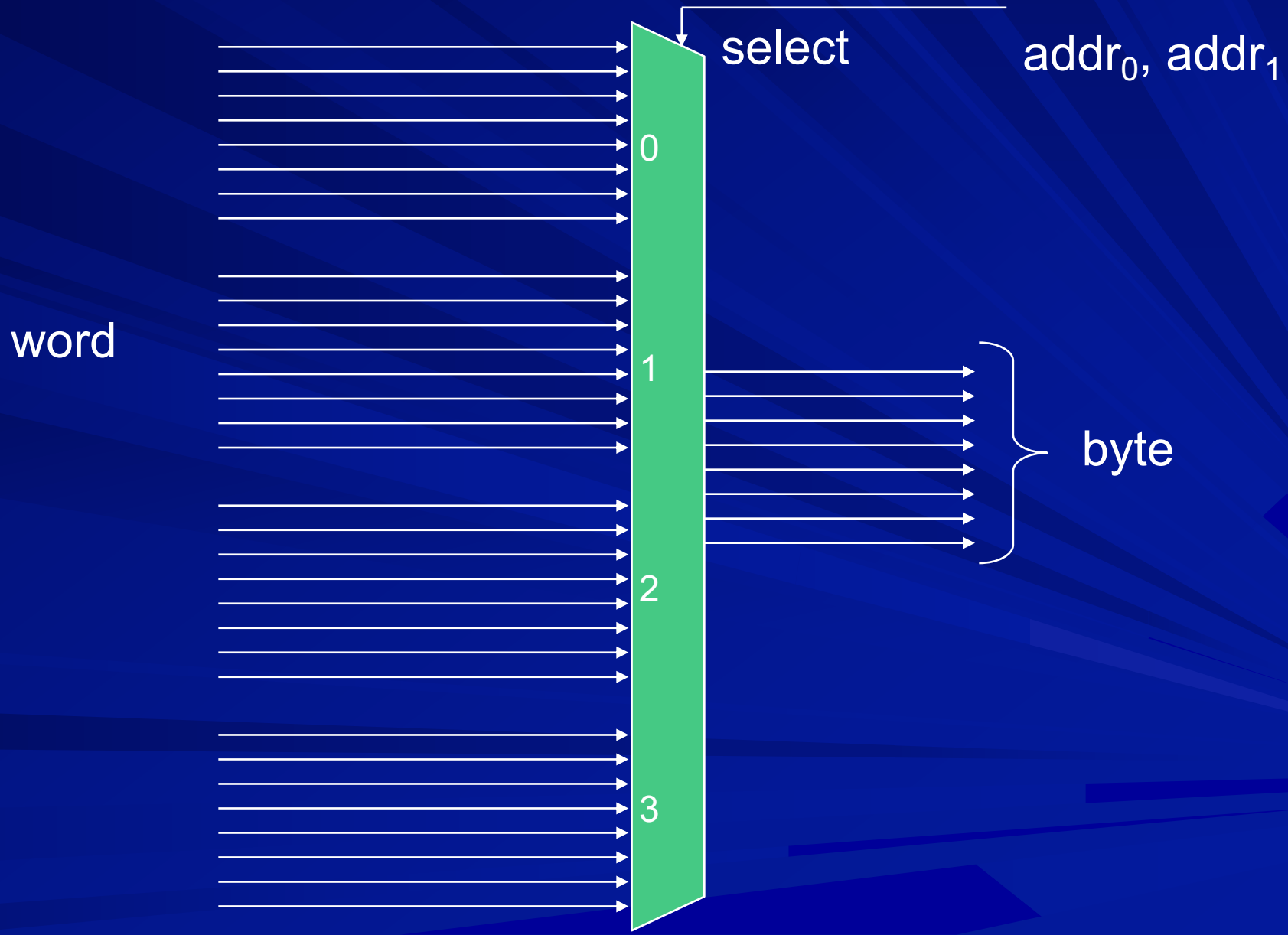
Pre/post increment/decrement

- Same operation for ALU whether pre or post
- Only the order / timings of address computation and memory access may differ

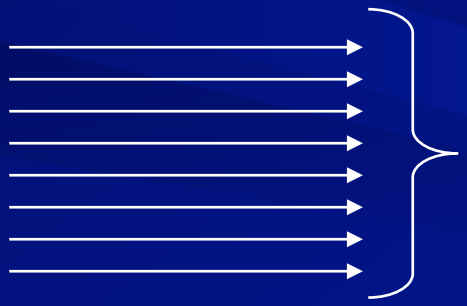
Auto increment/decrement

- Same operation for ALU whether auto or not
- Only the updation of base register may or may not take place

Word / half-word / byte transfer

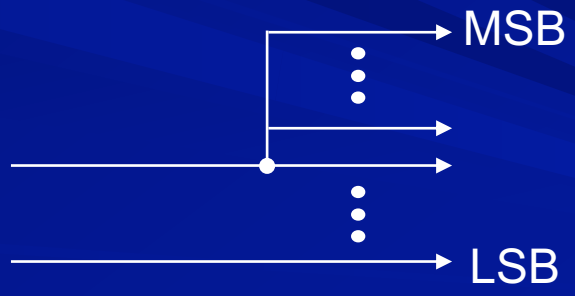


Word / half-word / byte transfer



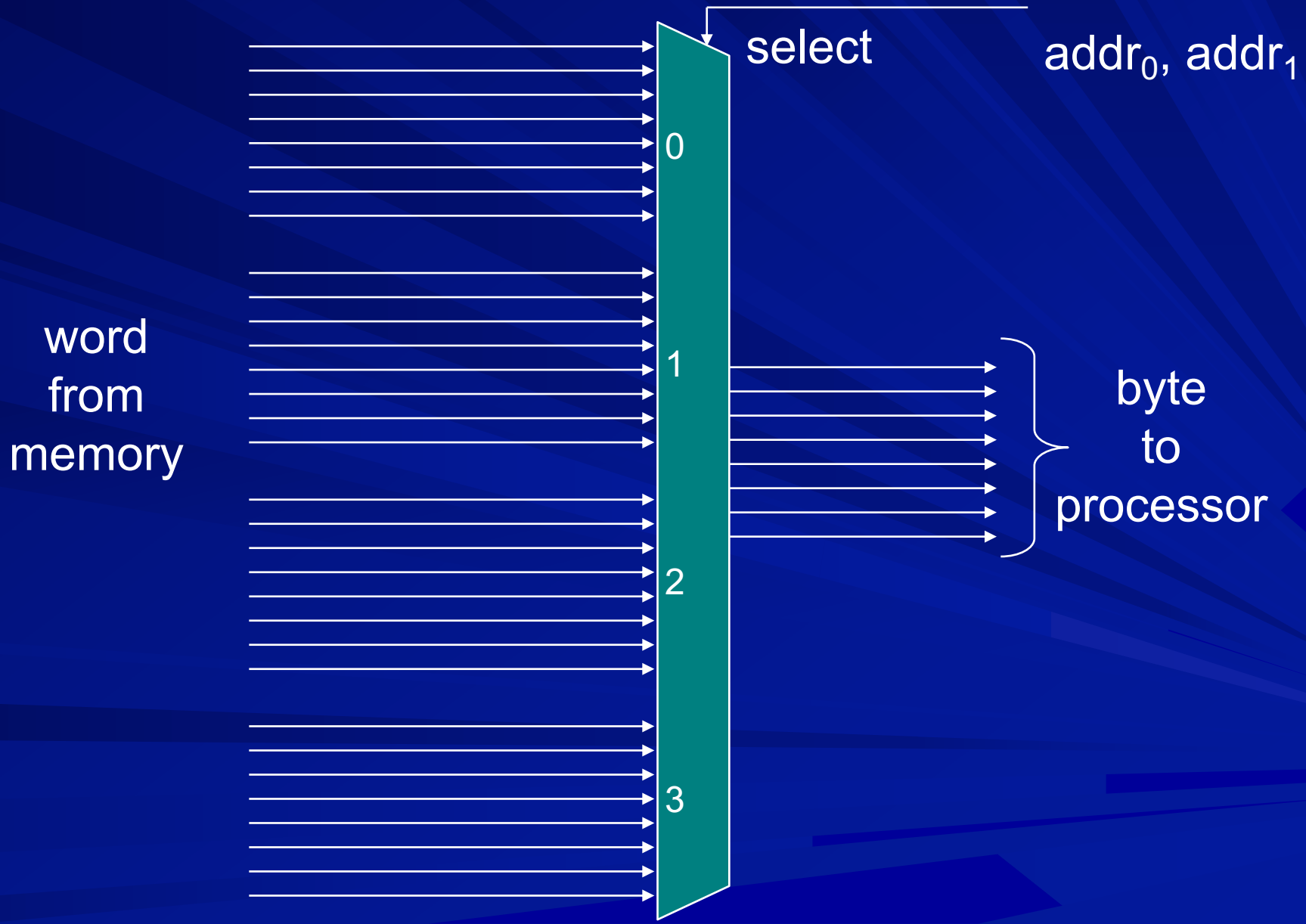
byte

zero extension

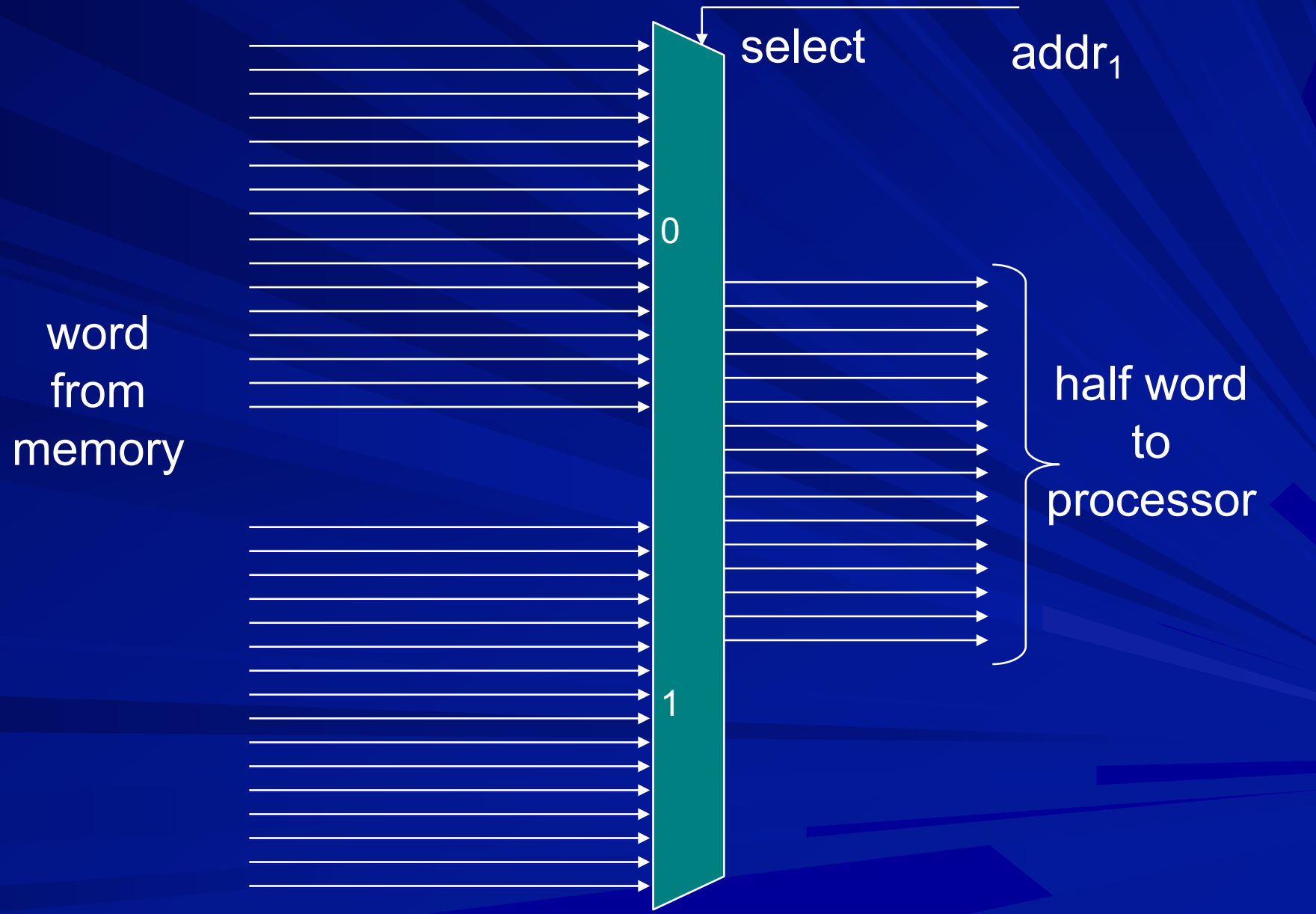


sign extension

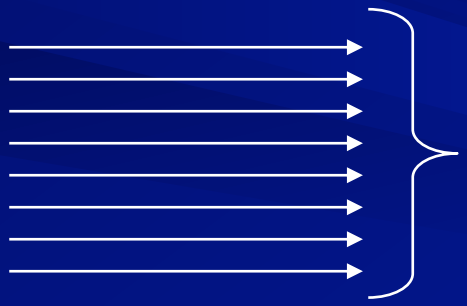
Selecting byte for ldrb / ldrsb



Selecting half word for ldrh / ldrsh

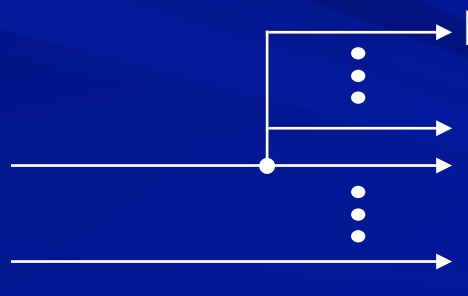


Extending from byte to word



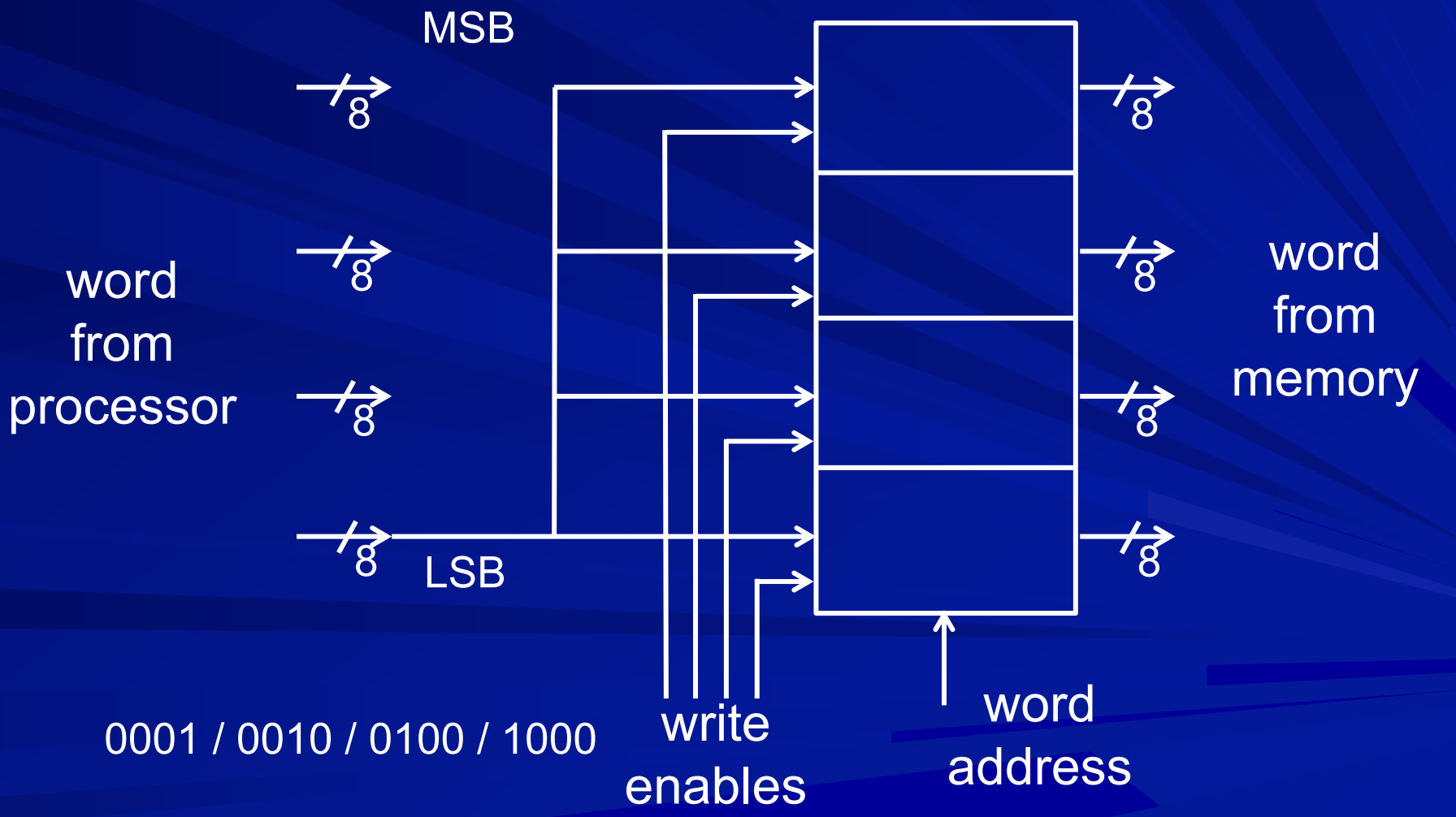
byte

zero extension

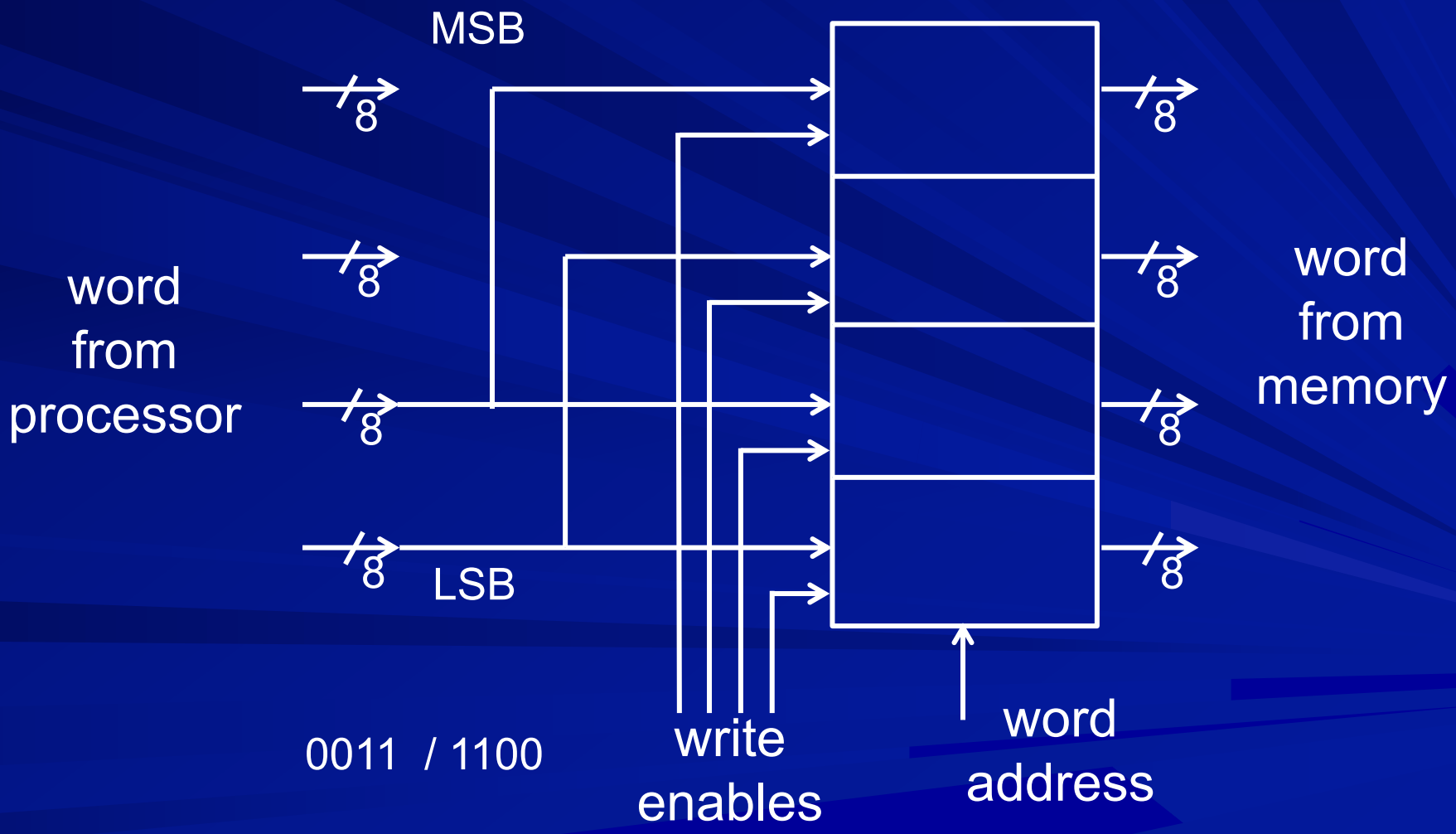


sign extension

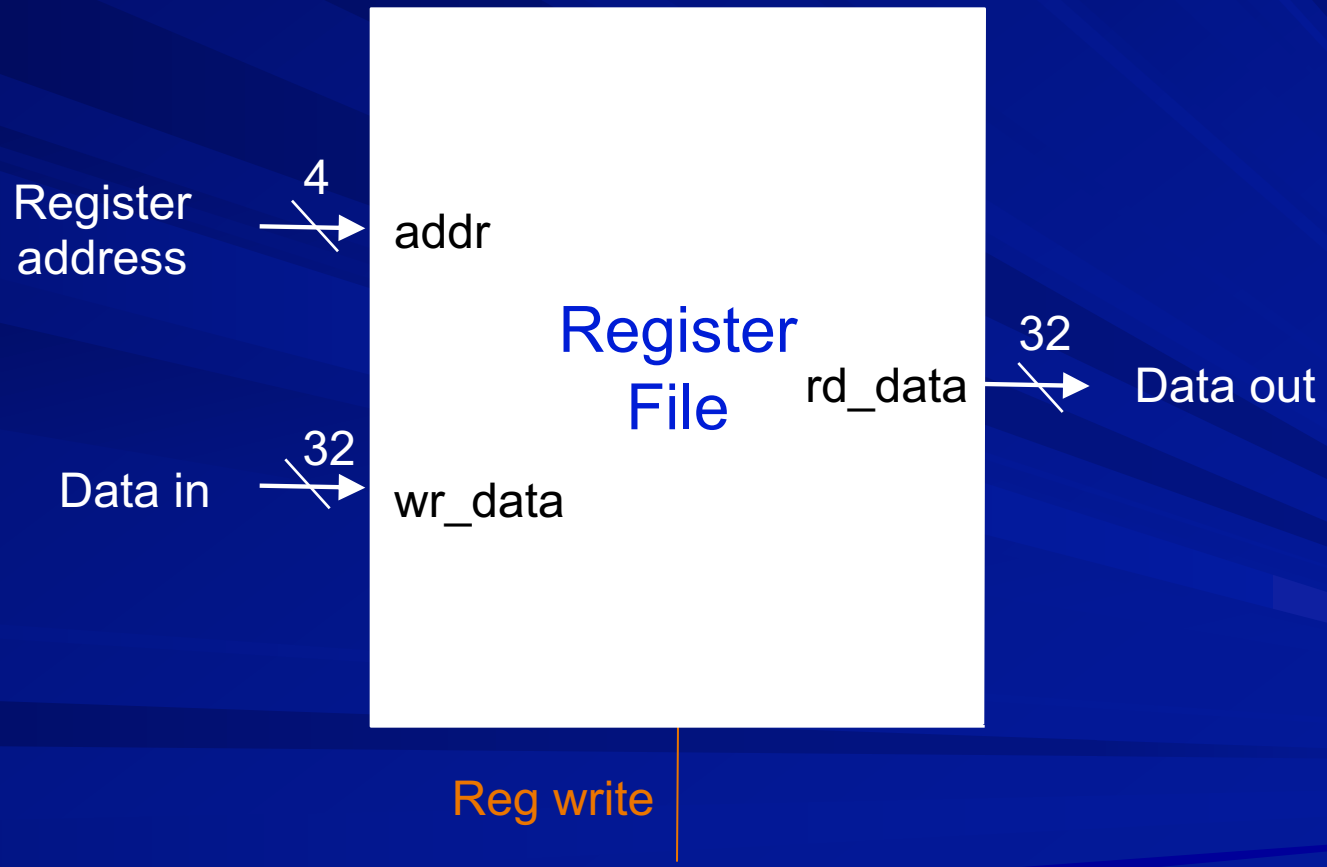
Writing a byte in memory



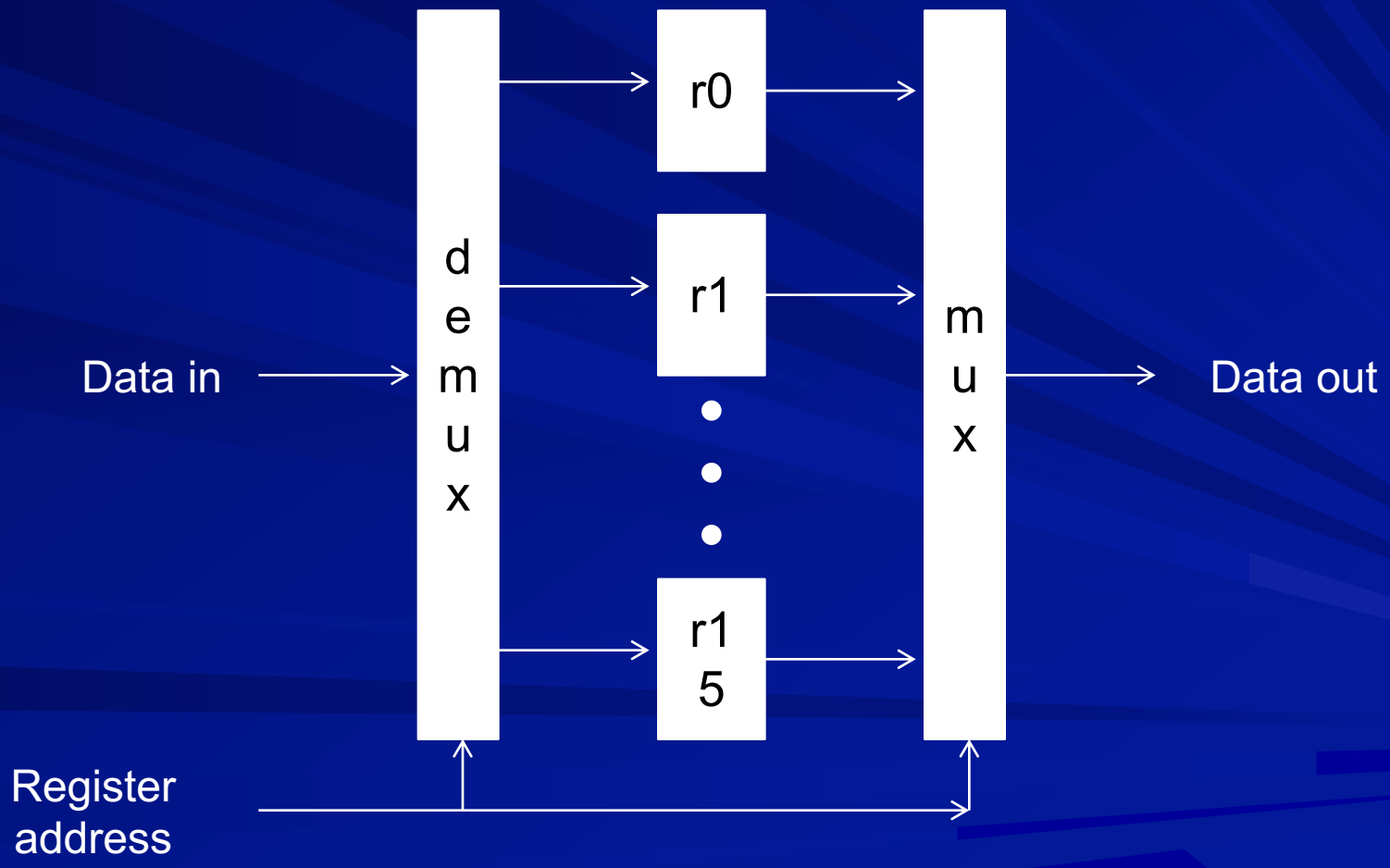
Writing a half word in memory



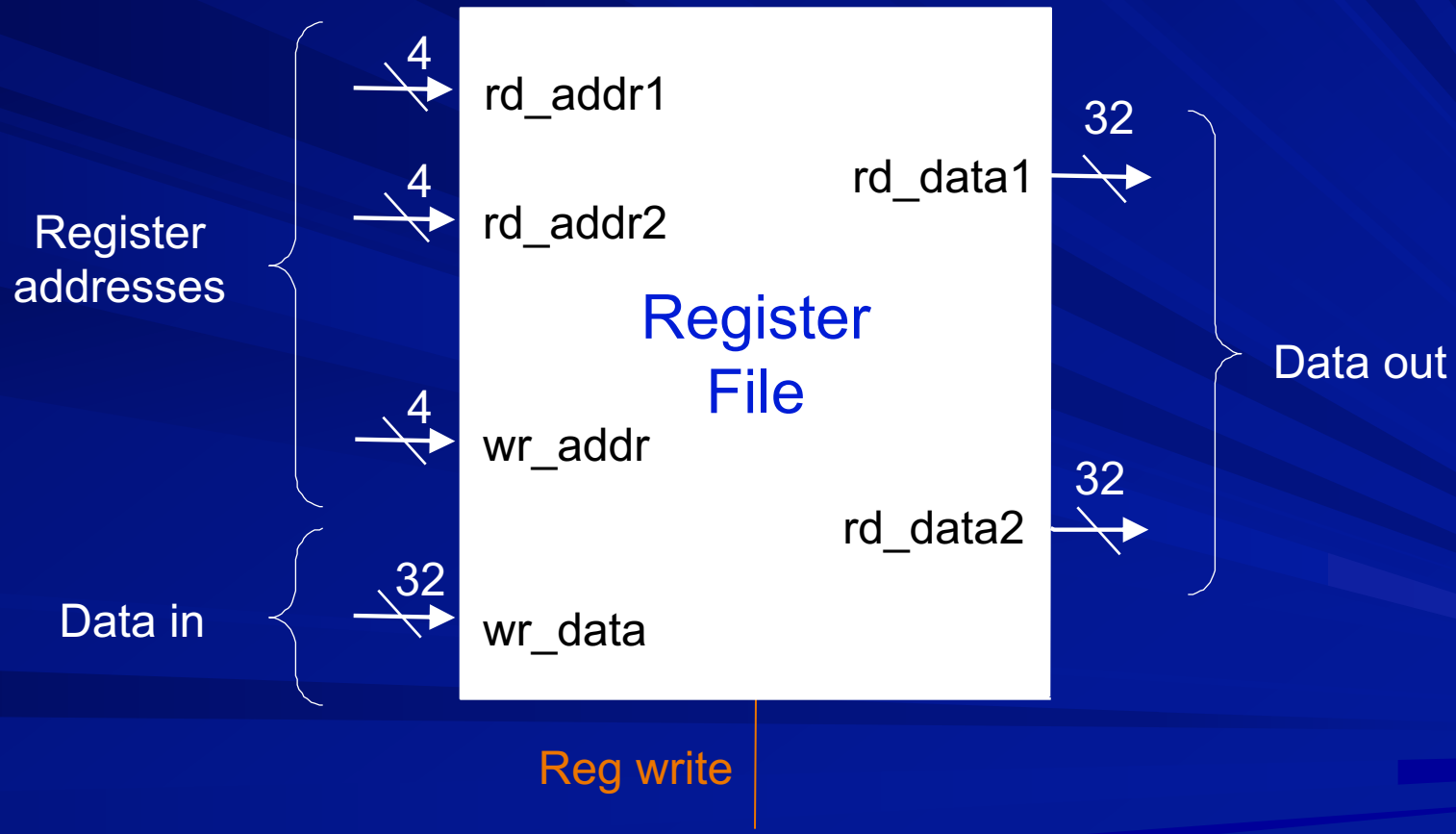
Register file



Register file



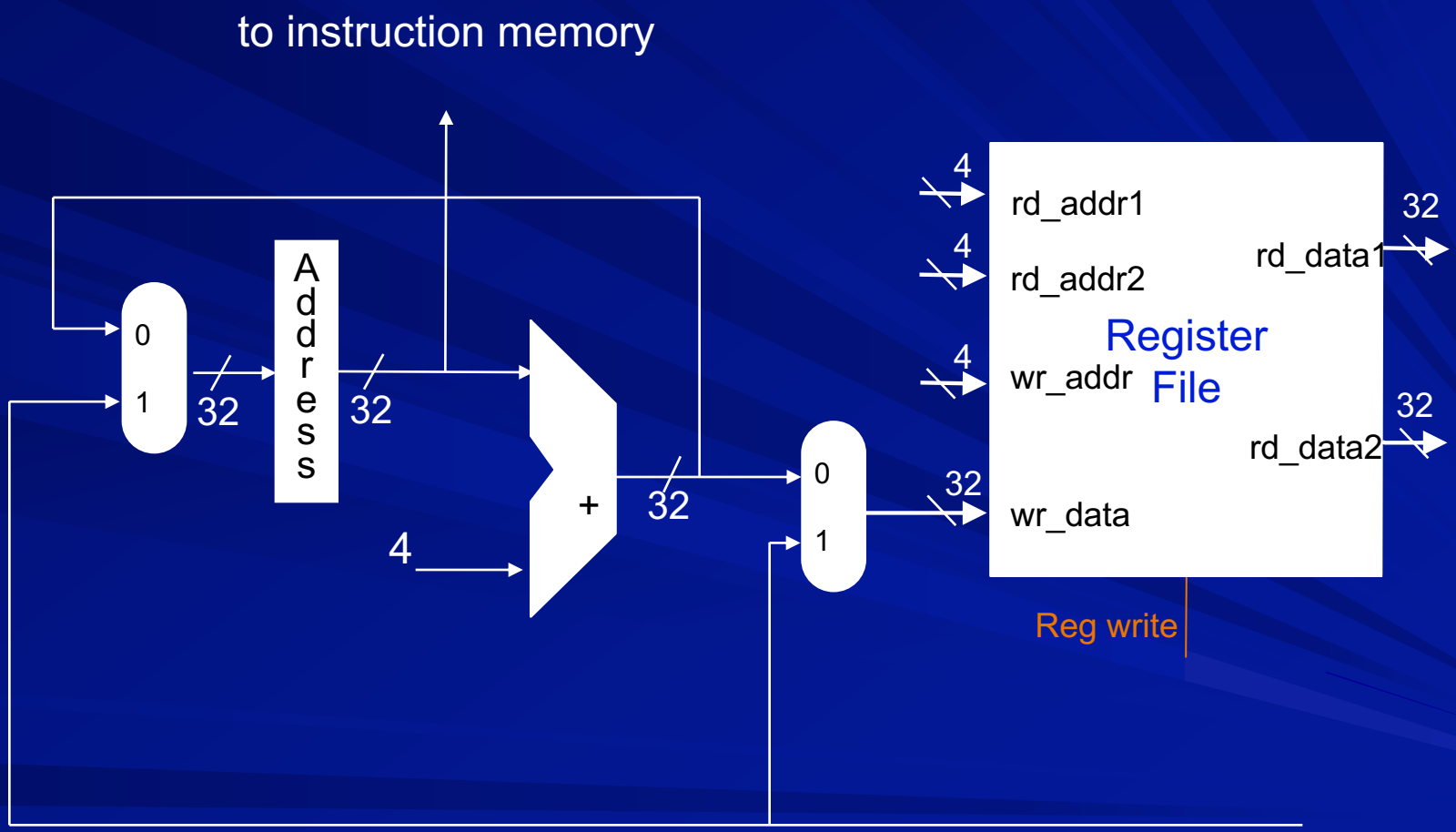
Register file (2R + 1W ports)



RF ports for ARM

- add r1, r2, r3, LSL r4
  requires 3 reads (r2, r3, r4) and 1 write (r1)
- ldr r1, [r2, r3]! and ldr r1, [r2], r3
  require 2 reads (r2, r3) and 2 writes (r1, r2)
- str r1, [r2, r3]! and str r1, [r2], r3
  require 3 reads (r1, r2, r3) and 1 write (r2)
- Additionally, 1 read + 1 write for r15 (PC) – all instructions read and write PC

Accessing PC



Thanks