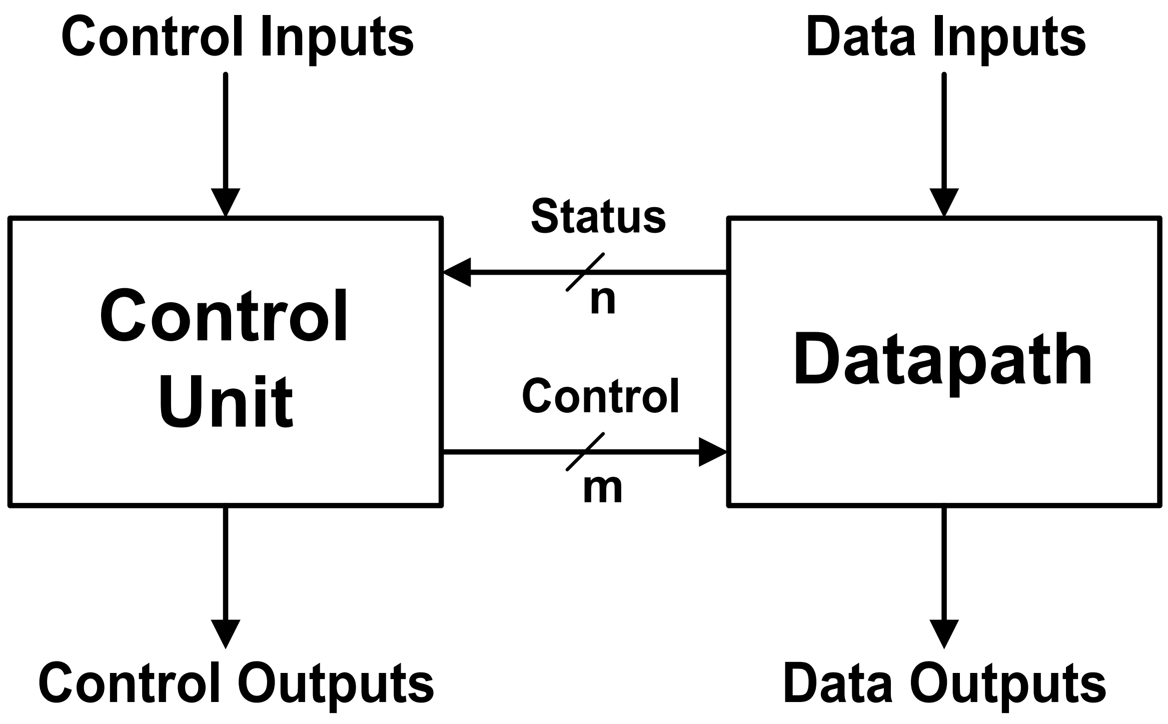
Introduction to Computer Organization

Large systems (e.g. a computer) are built in a modular, hierarchical structure using the basic methods of combinational and sequential design.

Most systems can be viewed as consisting of a datapath and a control unit .



Datapath

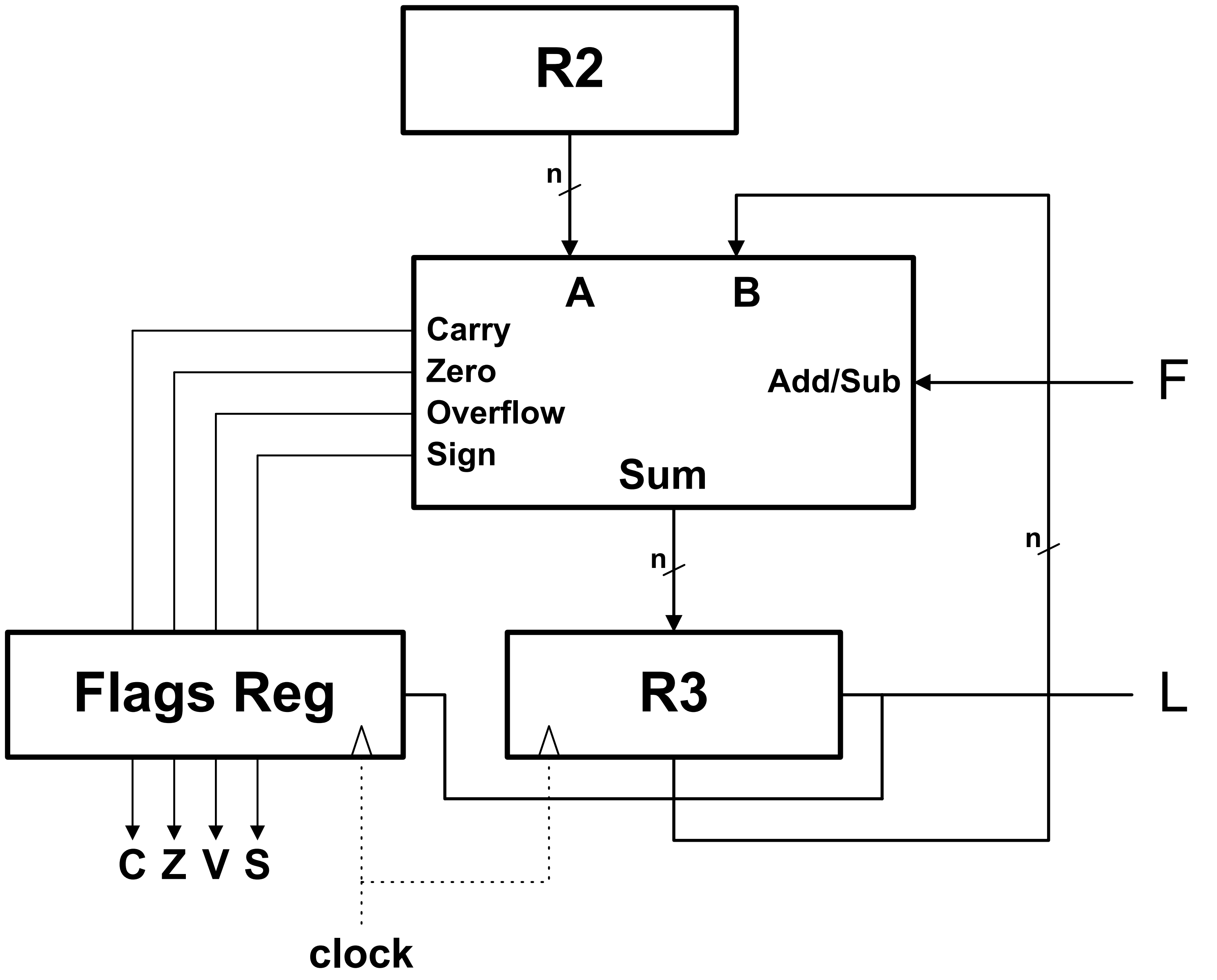
Datapaths may be defined in terms of their registers and the transfer of data among these registers.

An elementary operation using one or more registers that can take place in a *single* clock pulse is called a microoperation.

Examples of microoperations:

- **Transferring data between registers**
- **Clearing, shifting, incrementing, decrementing or negating a register**
- **Arithmetically combining registers (e.g. add, subtract …)**
- **Logically combining registers (e.g. AND, OR, XOR)**

A simple arithmetic unit



How does a computer execute programs?

A datapath is controlled by the input of a control word that defines a microoperation.

Each microoperation is followed by a clock pulse, to execute the microoperation.

A series of microoperations is called a microprogram.

Microprograms are stored in on-chip ROM called the "control ROM" and are fixed by the chip designer.

A machine-language instruction for a computer is defined by a microprogram which may take from 1 to several microoperations.

The sequence of events

C program compiled

↓

Assembly language program (for the target processor)

↓

binary codes (called opcodes or machine code) representing each assembly language instruction

↓

address in the control ROM containing the microprogram for that instruction

↓

control word(s) applied to the datapath on each clock pulse

Arithmetic Microoperations

Examples:

Add	$R1 \leftarrow R1 + R2$
1's Comp	$R4 \leftarrow R4'$
2's Comp	$R3 \leftarrow R3' + 1$
Subtract	$R1 \leftarrow R1 + R2' + 1$
Increment	$R5 \leftarrow R5 + 1$
Decrement	$R5 \leftarrow R5 - 1$

Simple instructions like ADD (+) need only a single microoperation (1 clock pulse).

More complex instructions like MUL (*) may use several clocks to execute a microprogram that implements a multiply algorithm.

Logical Microoperations

Examples:

AND	$R1 \leftarrow R1 \wedge R2$
OR	$R4 \leftarrow R4 \vee R27$
NOT	$R3 \leftarrow R3'$
XOR	$R1 \leftarrow R1 \oplus R2$

Shift Microoperations

Examples:

Shift left $R3 \leftarrow$ shl R3
Shift right $R4 \leftarrow$ shr R4
Rotate left $R3 \leftarrow$ rol R3
Rotate right rotate $R1 \leftarrow$ ror R1
Arithmetic shift left $R3 \leftarrow$ asl R3
Arithmetic shift right $R2 \leftarrow$ asrR2

Register Transfer Notation

The basic operation is designated:

$$R_d \leftarrow R_s$$

where R_s is the source and R_d is the destination register.

Note that R_s remains unchanged.

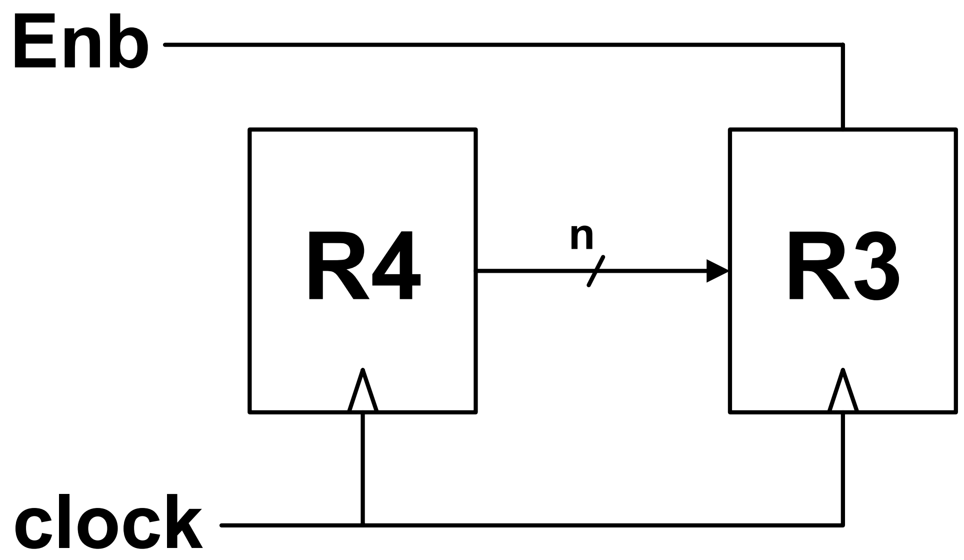
A conditional transfer is the more usual case:

$$\text{if } (Enb = 1) \text{ then } R3 \leftarrow R4$$

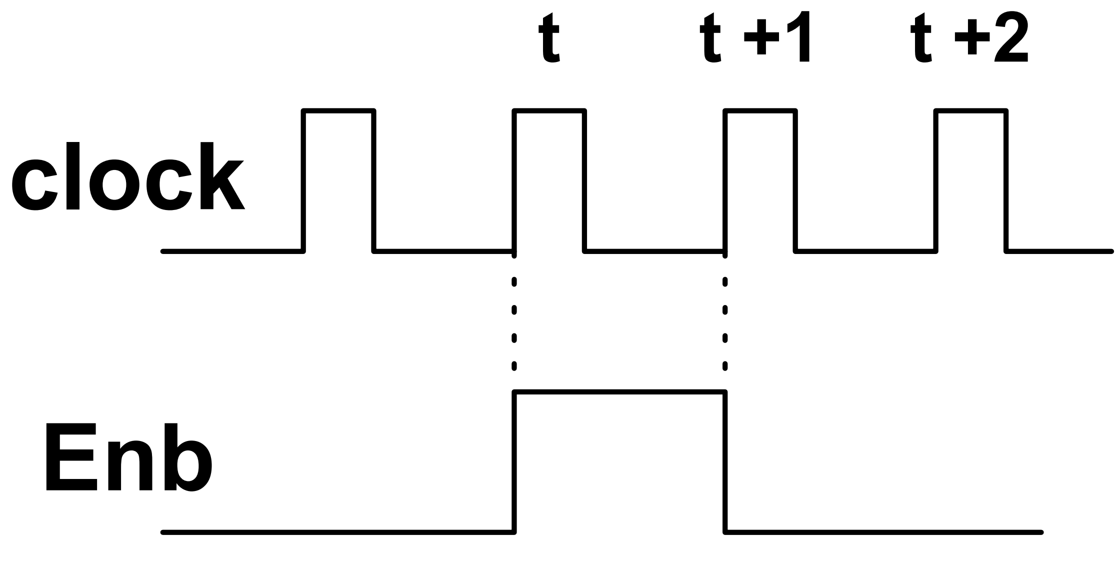
or more concisely:

$$Enb\text{: } R3 \leftarrow R4$$

In the hardware, the transfer is assumed to occur in response to a system clock pulse.



It is also assumed that the control input (eg. Enb) is synchronized to the system clock:



Multiple microoperations may occur on the same clock:

Enb: R3 $\leftarrow$ R4, R1 $\leftarrow$ R2

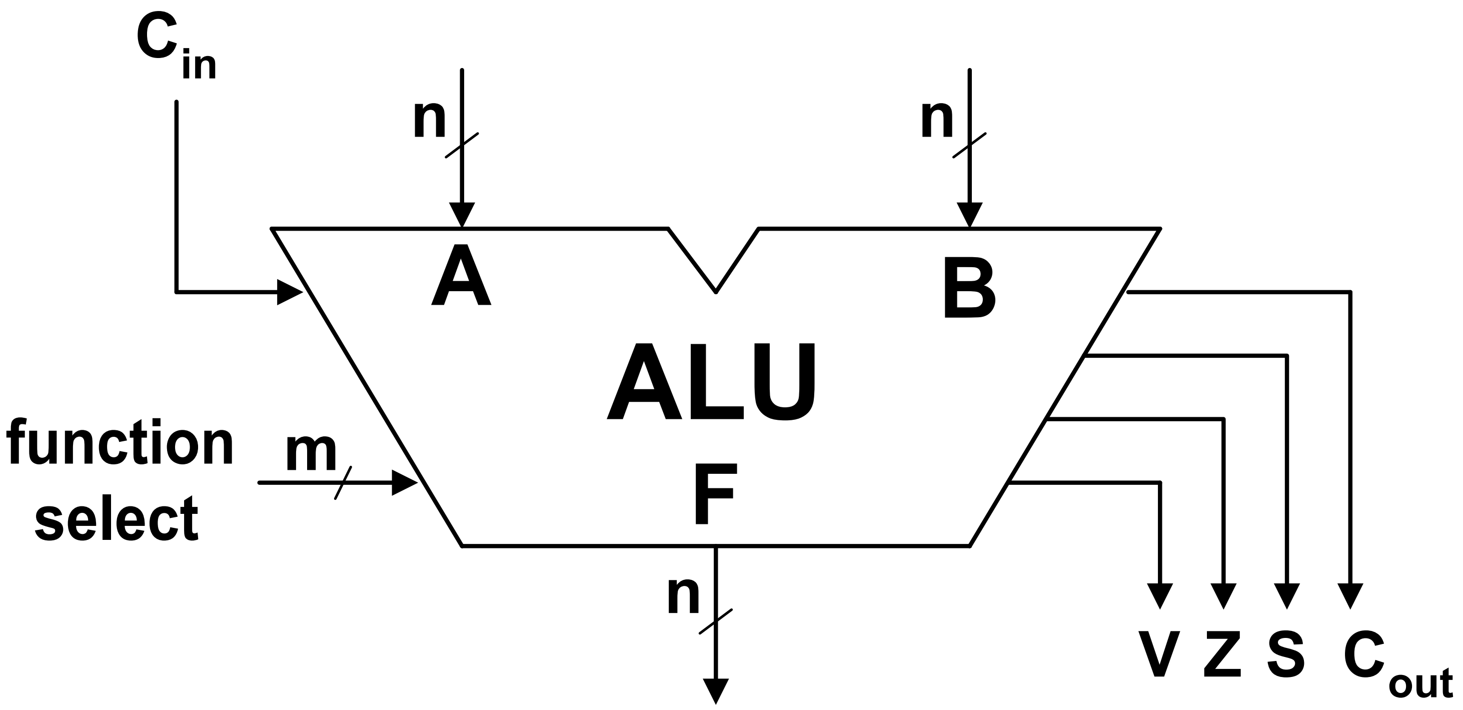
We can specify a portion of a register and constant data. For example, set the MSB of R3 to 0:

$$\text{R3(7)} \leftarrow 0$$

or, set the 4 MSBs of R3 to 1:

$$\text{R3(7:4)} \leftarrow 1$$

Arithmetic/Logic Unit (ALU)



V – overflow   Z – zero
S – sign (sometimes written as N)
C_{out} – carry out (sometimes written as C)

The basis for the ALU design is a parallel adder ($G = X + Y + C_{in}$). By designing logic that operates on one or both of the data inputs to the adder, a variety of functions can be implemented.

Function table for an ALU

Select S1 S0	Input Y	$G = X + Y + C_{in}$ $C_{in} = 0$	$C_{in} = 1$
0 0	0's	A (transfer)	A + 1 (increment)
0 1	B	A + B (add)	A + B + 1
1 0	B′	A + B′	A + B′ + 1 (subtract)
1 1	1's	A - 1 (decrement)	A (transfer)

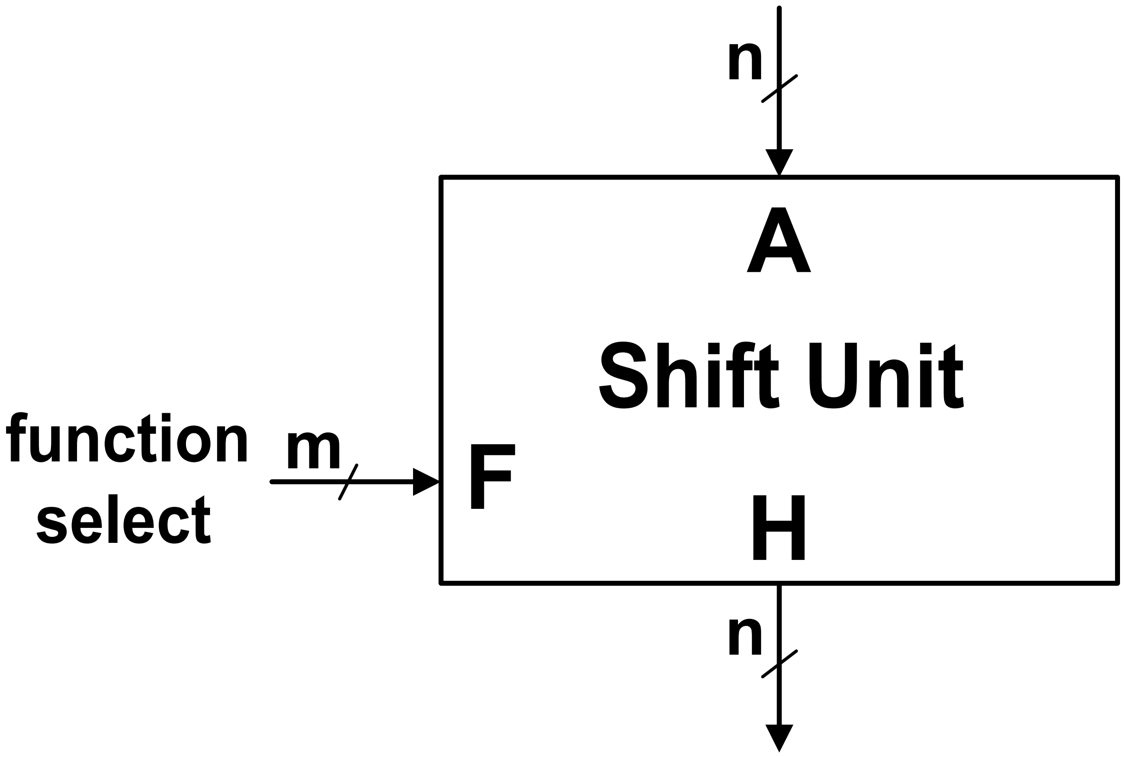
Logic Unit

The design of the logic functions should be integrated with that of the arithmetic function.

Conceptually, they can be viewed as separate units that are combined with an additional selection line.

Shift Unit

Block symbol:



Note: the shift unit is a combinational circuit in the datapath and does not require a clock pulse.

Why? Because the same combinational logic can be re-used by multiple source/ destination registers

A Simple Example

F (m=1)	Output H
0	shl (A)
1	shr (A)

Multiplying by other than powers of 2 can be achieved in multiple microoperations. For example:

R4 ← shl (R5), R6 ← R5 + R4

loads R6 with 3 x R5.

Another More Complex Example

F (m=3)	Output H
0 0 0	A
0 0 1	shl (A)
0 1 0	shr (A)
0 1 1	rol (A)
1 0 0	ror (A)
1 0 1	asr (A)
1 1 0	rlc (A)
1 1 1	rrc (A)

rlc(A) – rotate left with carry
rrc(A) – rotate right with carry

Note: mnemonics vary from one processor (microcontroller) to another!

Barrel Shift Unit

A barrel shift unit is a combinational circuit that rotates the input bits by the number of bit positions specified by the input function lines.

Note that a left barrel shift unit can generate all right rotations. In general in a n-bit barrel shift unit, m positions of left rotation is the same as n - m bits of right rotation.

Design of Datapath

The design of the datapath determines the fundamental "architecture" or "organization" of the computer.

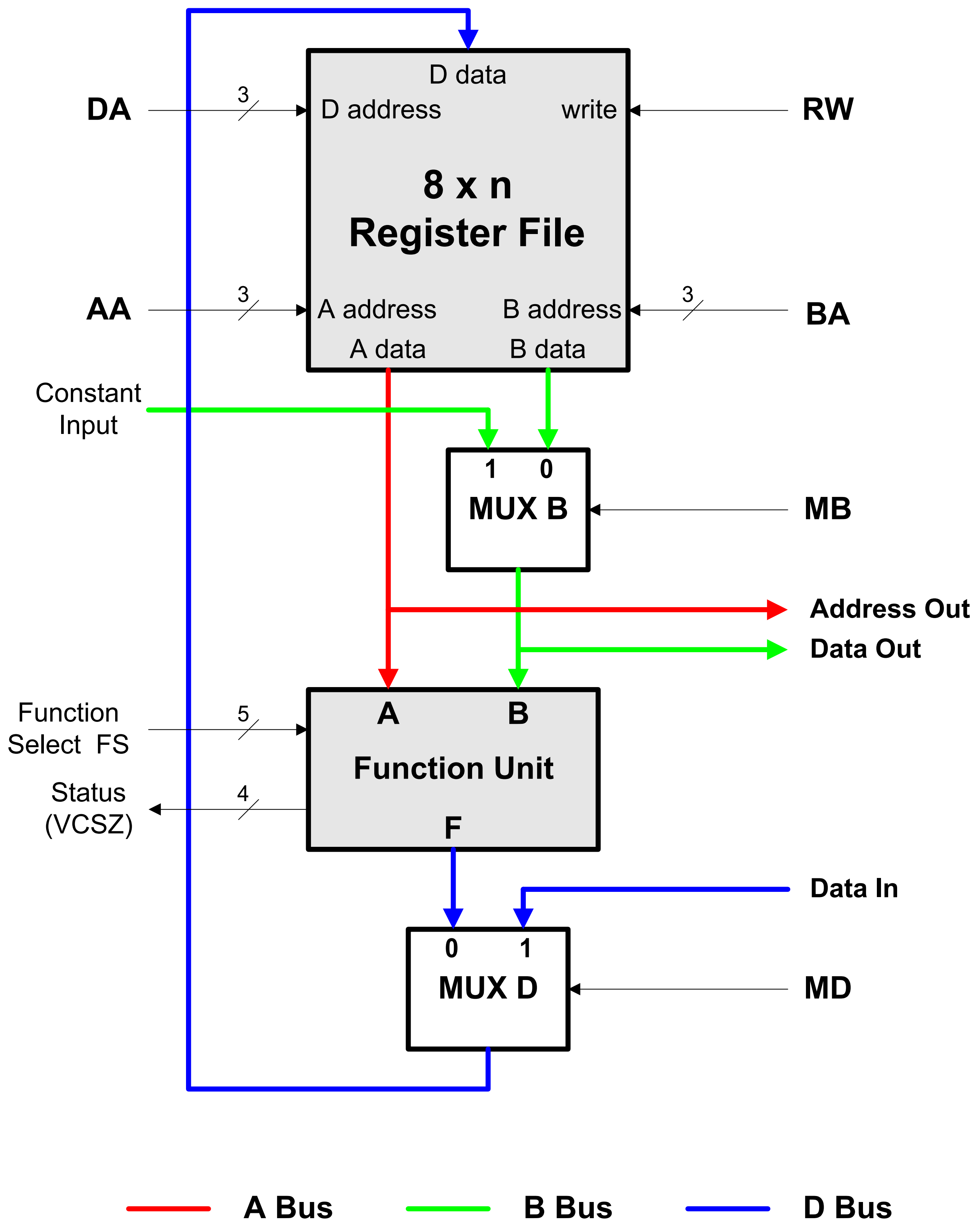
The datapath contains the processor data register set and defines the functions that may be performed on these registers.

It also has an interface to external data memory.

The microoperation performed at each clock pulse is specified by a control word:

Control Word

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DA			AA			BA			MB	FS					MD	RW


D data
DA
3
D address
write
RW
8 x n
Register File
AA
3
A address
B address
3
BA
A data
B data
Constant
Input
1
0
MUX B
MB
Address Out
Data Out
Function
Select FS
5
A
B
Function Unit
Status
(VCSZ)
4
F
Data In
0
1
MUX D
MD
A Bus
B Bus
D Bus

Function Unit

FS	Operation
00000	$F = A$ (transfer)
00001	$F = A + 1$ (increment)
00010	$F = A + B$ (add)
00011	$F = A + B + 1$ (add & increment)
00100	$F = A + B' = A - B - 1$ (subtract & decrement)
00101	$F = A + B' + 1 = A - B$ (subtract)
00110	$F = A - 1$ (decrement)
00111	$F = A$ (transfer)
01000	$F = A \wedge B$ (AND)
01010	$F = A \vee B$ (OR)
01100	$F = A \oplus B$ (XOR)
01110	$F = A'$ (NOT)
10000	$F = \text{shr } A$ (shift right)
10001	$F = \text{shl } A$ (shift left)

Register Addresses

AA, BA or DA	Register
0 0 0	R0
0 0 1	R1
0 1 0	R2
0 1 1	R3
1 0 0	R4
1 0 1	R5
1 1 0	R6
1 1 1	R7

Examples of Microoperations:

Microoperation	Control Word (17 bits) DA AA BA MB FS MD RW
R1 ← R2 – R3	
R4 ← shl R6	
R5 ← data in	
R7 ← R7 + 5	

Pipelined Datapath

The performance or “throughput” of the datapath may be improved using the concept of “pipelining.”

A pipeline organization is created by inserting registers in the datapath to hold intermediate results.

Two pipeline registers may be used to divide the datapath into three sections:

Operand fetch (OF)
Execute (EX)
Write-back (WB)

The pipeline registers are clocked simultaneously.

Pipeline Example

Consider a series of 7 consecutive microoperations operating in a three-stage pipeline:

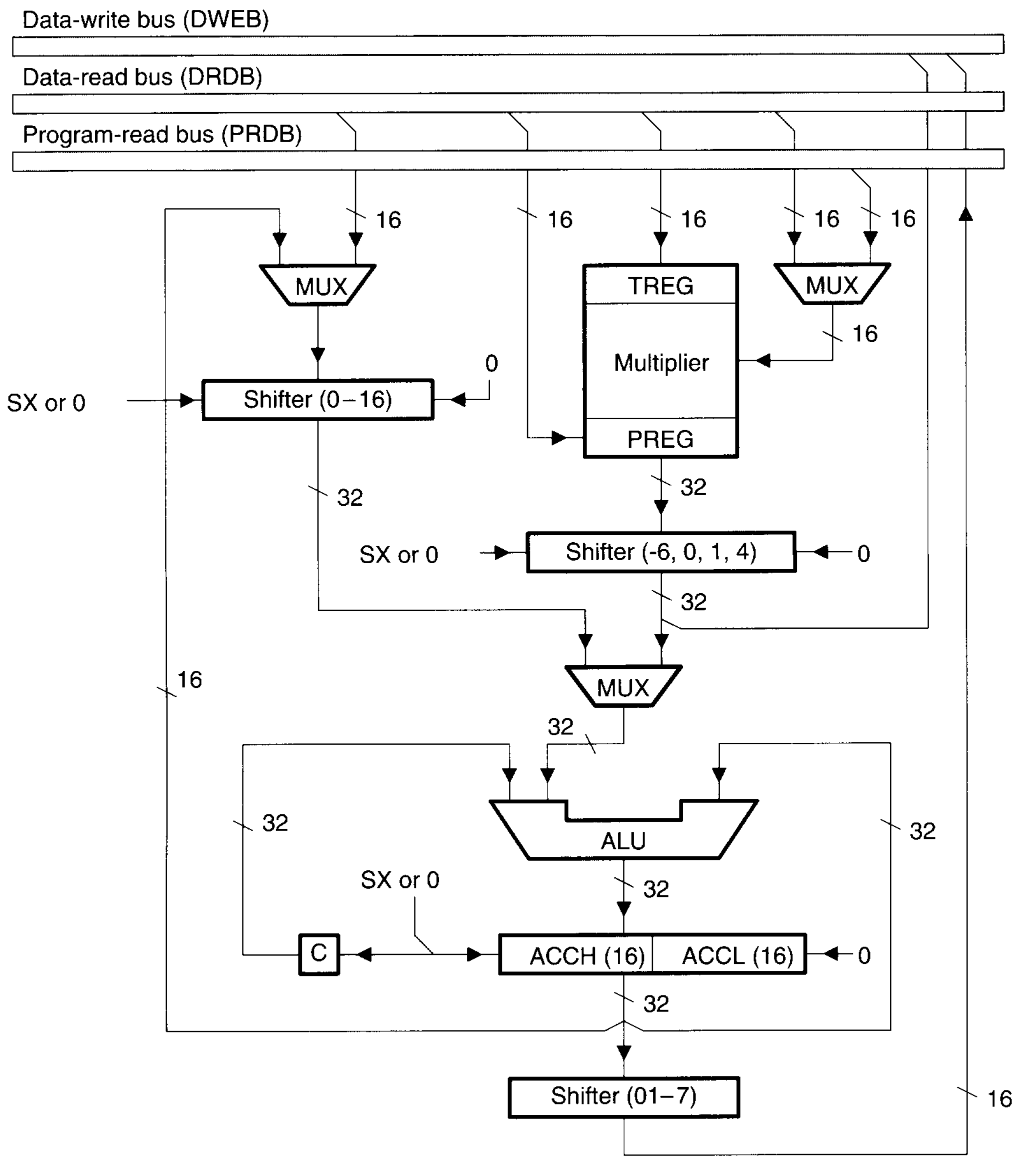
clock pulse #

	1	2	3	4	5	6	7	8	9
1	OF	EX	WB						
2		OF	EX	WB					
3			OF	EX	WB				
4				OF	EX	WB			
5					OF	EX	WB		
6						OF	EX	WB	
7							OF	EX	WB

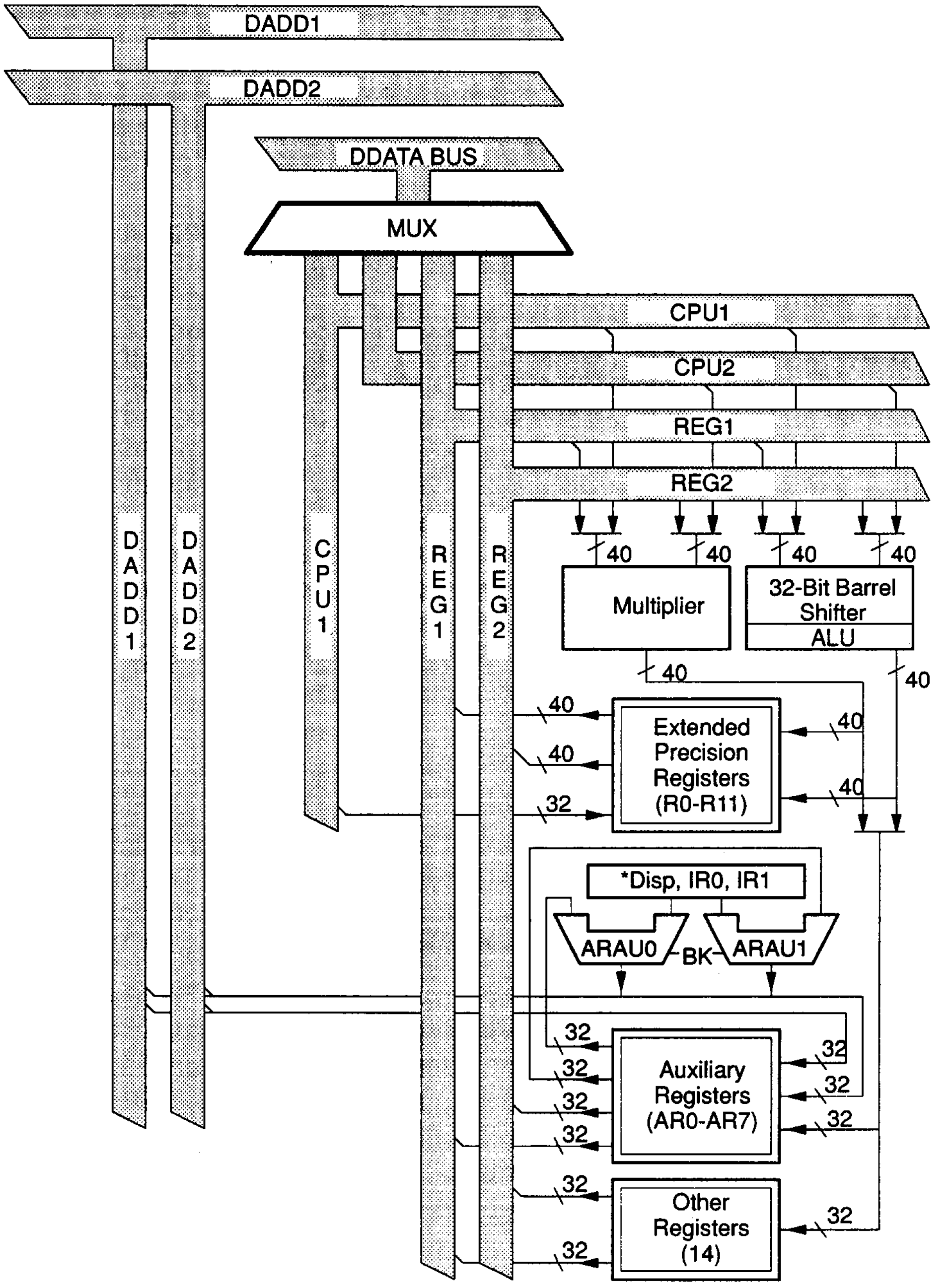
$\uparrow$

microoperations #

TI C2XX DSP


Data-write bus (DWEB)
Data-read bus (DRDB)
Program-read bus (PRDB)
16
16
16
16
16
MUX
TREG
MUX
16
Multiplier
0
SX or 0
Shifter (0–16)
PREG
32
32
SX or 0
Shifter (-6, 0, 1, 4)
0
32
16
MUX
32
32
ALU
32
SX or 0
32
C
ACCH (16)
ACCL (16)
0
32
Shifter (01–7)
16

TI 'C40 DSP


DADD1
DADD2
DDATA BUS
MUX
CPU1
CPU2
REG1
REG2
DADD1
DADD2
CPU1
REG1
REG2
40
40
40
40
Multiplier
32-Bit Barrel Shifter
ALU
40
40
40
40
40
40
32
Extended Precision Registers (R0-R11)
*Disp, IR0, IR1
ARAU0
BK
ARAU1
32
32
32
32
32
32
32
Auxiliary Registers (AR0-AR7)
32
32
32
Other Registers (14)