Basic Computer Architecture

- Computers c
  - Datapath
  - Control unit
- It is designed to implement a particular instruction set.
- The individual instructions are the engineering equivalent of the mathematician's
  - **z=f(x,y)**

f z x,y

Opcode – Destination – Operands

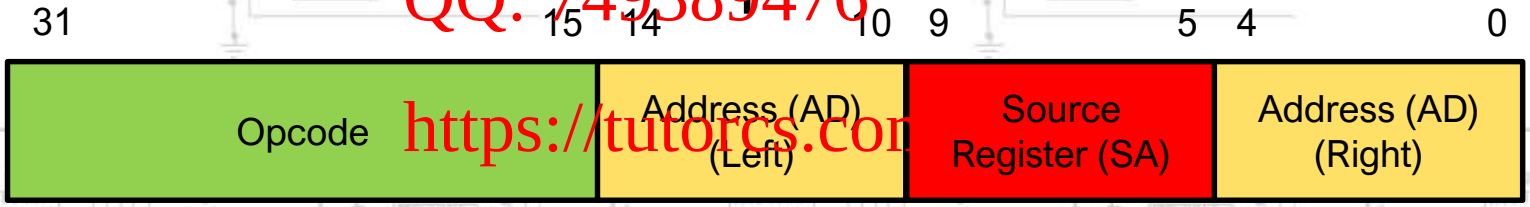
- OPCODE
  - Selects the function
- DESTINATION
  - Is nearly always a datapath register
- OPERANDS
  - Usually come from datapath register

Instruction Format Examples

Register

Immediate

Jump and Branch



Instruction Formats

- Where DR, SA ∧ SB point to processor registers in the datapath
- But Operand is itself an immediate operand

Data and Instructions in Memory

Example for a processor, with 7, 3, 3, 3 bits

Decimal address	Memory contents	Decimal opcode	Other specified fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2 SB:3	R1 ← R2 − R3
35	0100000 000 100 101	32 (Store)	SA:4 SB:5	M [R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2 SA:7 OP:3	R2 ← R7 + 3
55	1100011 101 110 100	96 (Branch on zero)	AD: 44 SA:6	If R6 = 0, PC ← PC − 20
70	0000000 011 000 000	Data = 192. After execution of instruction in 35, Data = 80.		

User View of Storage

Register File
32 x 32

gram Counter (PC)

Instruction Memory
2^{32} x 16

Data Memory
2^{32} x 16

Memory Module [entity]

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity memory is -- use unsigned for memory address
    Port ( address : in unsigned std_logic_vector(31 downto 0);
           write_data : in std_logic_vector(31 downto 0);
           MemWrite, MemRead : in std_logic;
           read_data : out std_logic_vector(31 downto 0));
end memory;
```

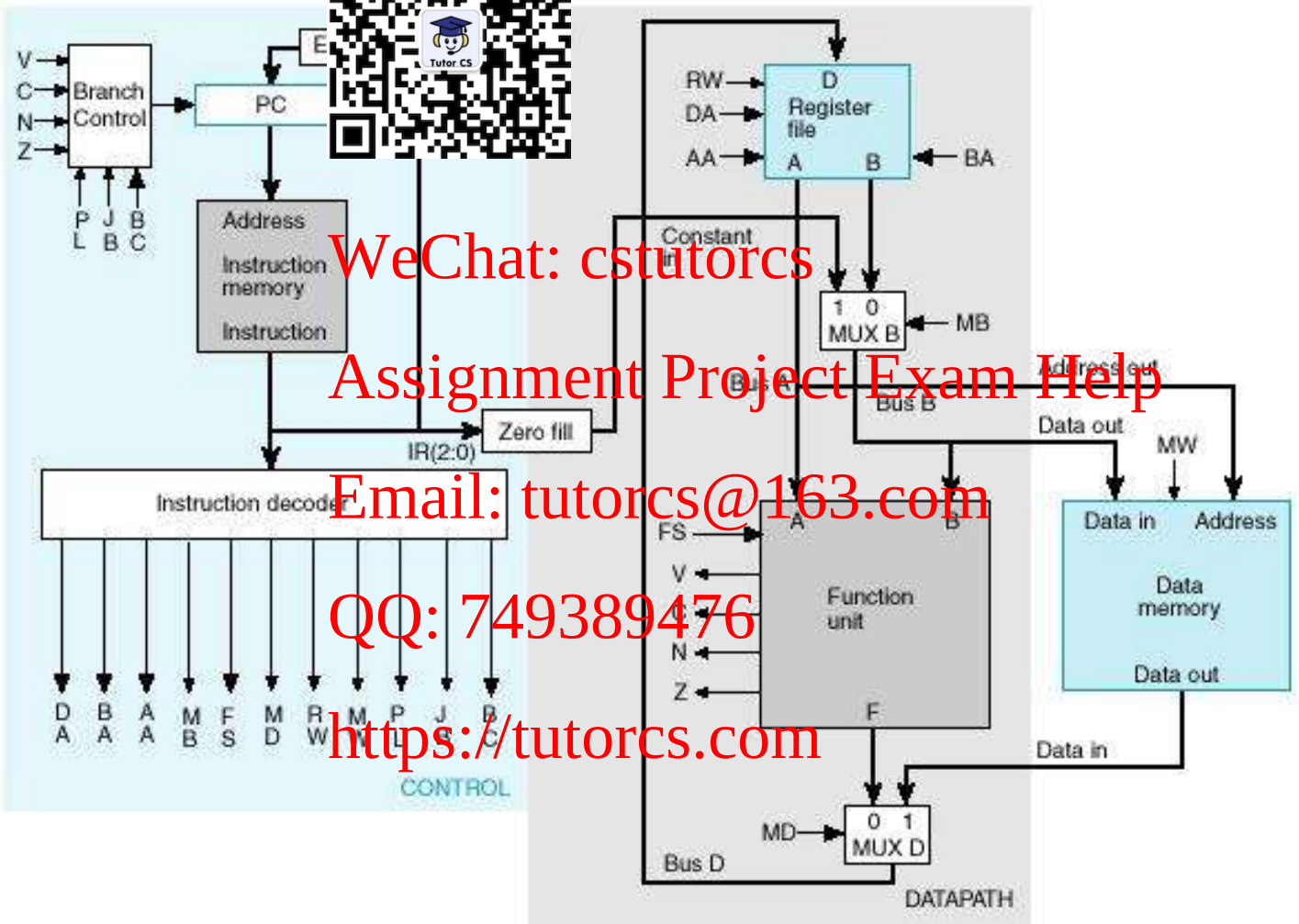
Memory Module [architecture]

```
architect        of memory is
-- we will        significant 9 bit of the address - array(0 to 512)
type mer        y(0 to 7) of std_logic_vector(31 downto 0);
-- define type, for memory arrays
begin
mem_process: process (address, write_data)
-- initialize data memory, X denotes hexadecimal number
variable data_mem : mem_array := (
X"00000000", X"00000000", X"00000000",X"00000000",
X"00000000", X"00000000", X"00000000",X"00000000");
variable addr:integer;
begin  -- the following type conversion function is in std_logic_arith
addr:=conv_integer(address(2 downto 0));
if MemWrite='1' then
data_mem(addr):= write_data;
elsif MemRead='1' then
read_data <= data_mem(addr) after 10 ns;
end if;
end process;
end Behavioral;
```

A Single-cycle Hardwired Control Unit

- We briefly consider a system with the simplest possible control unit.
- The control unit:
  - Maps each OP CODE to a single datapath operation.
- Instructions are fetched from an instruction memory
- This is what all present systems with separate instruction and data code do.

Single-Cycle Computer



Branch

- Instruction Bit
  - Jump occurs
- Instruction Bit13=0
  - Conditional Branch occurs
  - Bit11, Bit10 and Bit9
    - Select the status bit

BC	Status Bit
000	C
001	N
010	V
011	Z
100	$\overline{C}$
101	$\overline{N}$
110	$\overline{V}$
111	$\overline{Z}$

Jump and Branch

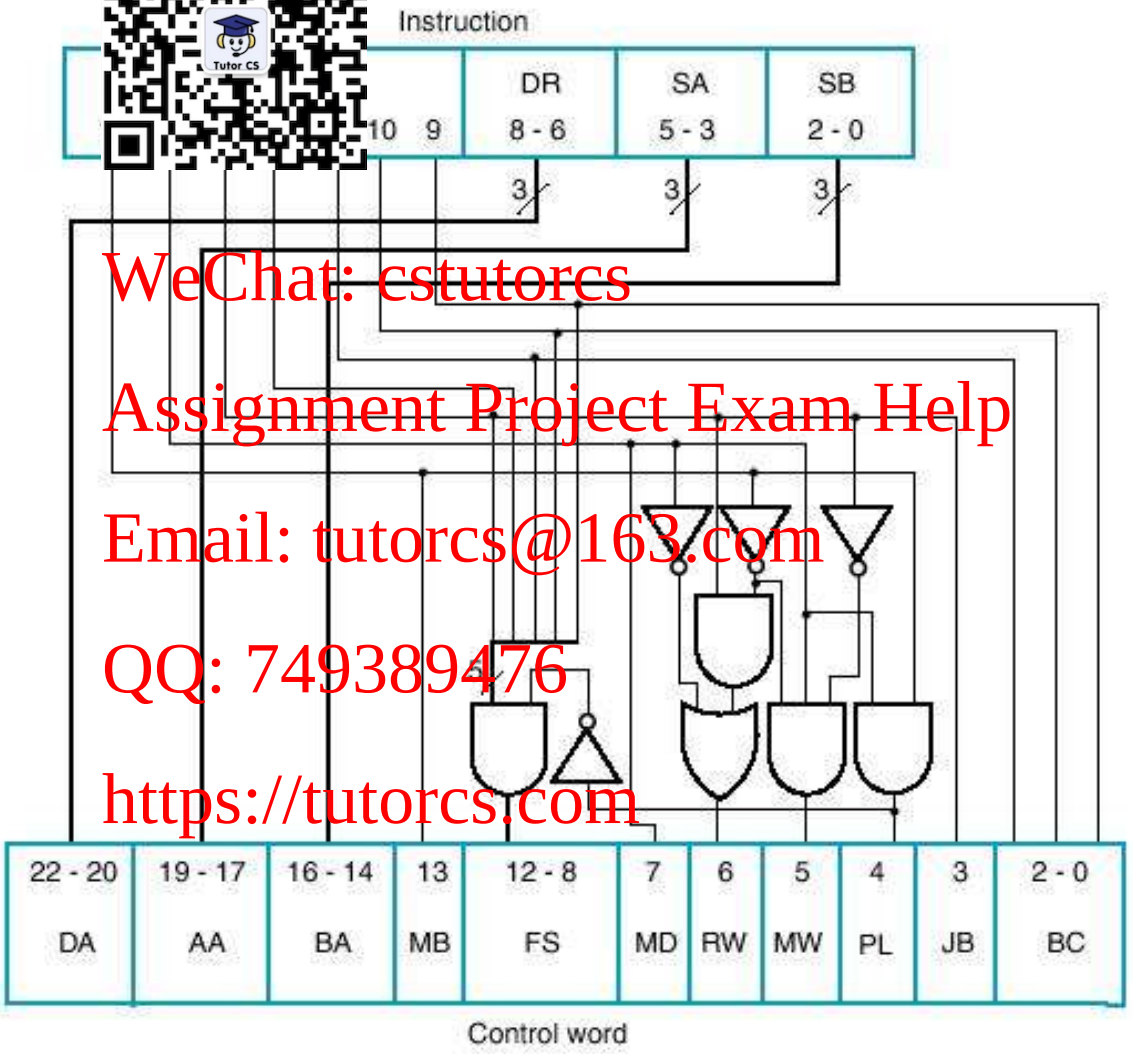
- PL=1
  - Jump or Branch, loading the PC
- PL=0
  - PC is incremented
- PL=1 ∧ JB=0
  - Jump
- PL=1 ∧ JB=1
  - Conditional branch

Truth Table BIT15 – BIT13

- The following operations classification helps with implementation of the instruction decoder

	Instruction Bits			Control Word Bits					
Instruction Function Type	**Bit 15**	**Bit 14**	**Bit 13**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**
ALU function using registers	0	0	0	0	0	1	0	0	X
Shifter function using registers	0	0	1	0	0	1	0	0	X
Memory write using register data	0	1	0	0	X	0	1	0	X
Memory read using register data	0	1	1	0	1	1	0	0	X
ALU operation using a constant	1	0	0	1	0	1	0	0	X
Shifter function using a constant	1	0	1	1	0	1	0	0	X
Conditional Branch	1	1	0	X	X	0	0	1	0
Unconditional Jump	1	1	1	X	X	0	0	1	1

Instruction Decoder



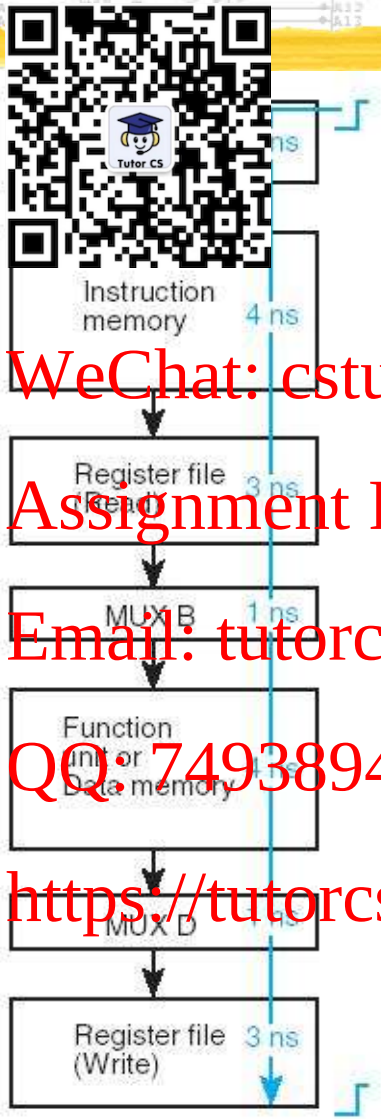
Single-Cycle Computer Instruction Example

Operation code	Symbolic name	Format		Function	MB	MD	RW	MW	PL	JB
1000010	ADI	Immediate	Add immediate operand	$R[DR] \leftarrow R[SA] + zf\ I(2:0)$	1	0	1	0	0	0
0110000	LD	Register	Load memory content into register	$R[DR] \leftarrow M[R[SA]]$	0	1	1	0	0	1
0100000	ST	Register	Store register content in memory	$M[R[SA]] \leftarrow R[SB]$	0	1	0	1	0	0
0011000	SL	Register	Shift left	$R[DR] \leftarrow sl\ R[SB]$	0	0	1	0	0	1
0001110	NOT	Register	Complement register	$R[DR] \leftarrow \overline{R[SA]}$	0	0	1	0	0	0
1100000	BRZ	Jump/Branch	If $R[SA] = 0$, branch to PC + se AD	If R[SA] = $0, PC \leftarrow PC + seAD$, If $R[SA] \neq 0, PC \leftarrow PC + 1$	1	0	0	0	1	0

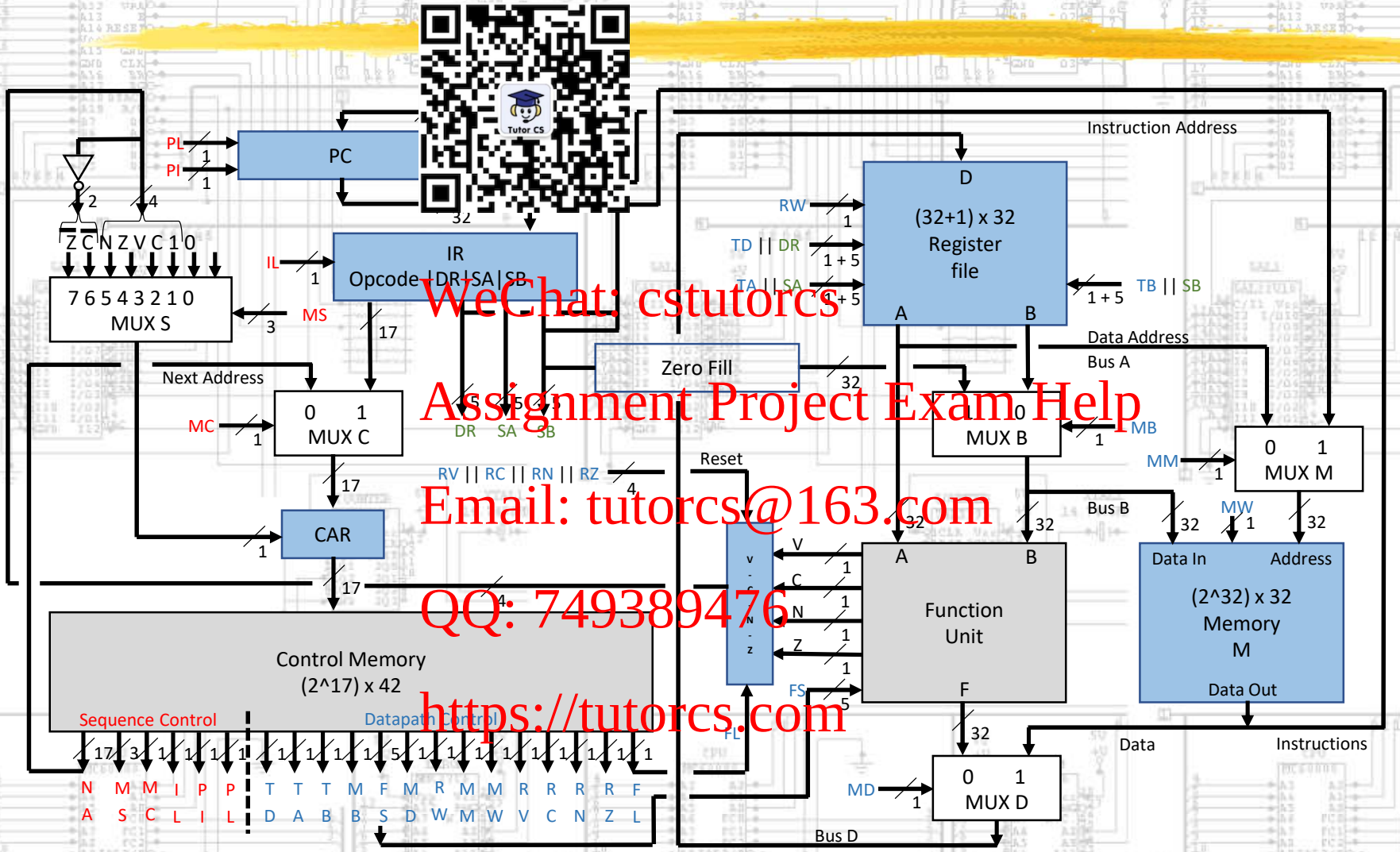
Single-cycle Problems

- A single-cycle control unit cannot implement:
  - more complex addressing modes
  - Composite functions
    - E.g. Multiplication
- A single-cycle control unit has long worst case delay path.
  - Slow clock.

Worst Case Delay

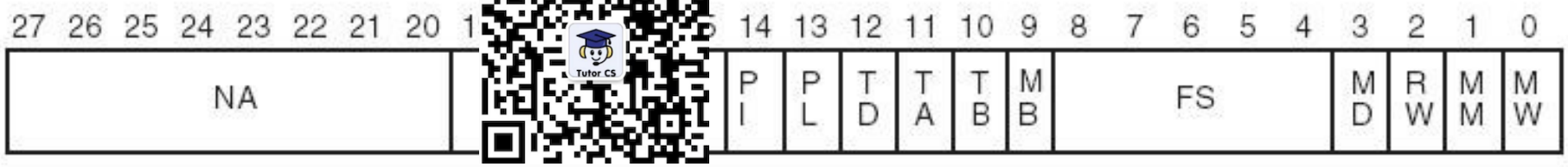


Multiple-Cycle Microprogrammed Computer



Microinstruction Format

Control Word Information for Datapath



TD Select	TA Select	TB Select	MB Select	Code	FS Function	Code	MD Select	RW Function	MM Select	MW Function	Code
R[DR]	*R*[SA]	*R*[SB]	Register	0	$F = A$	00000	Fn Out	No write (NW)	Address	No write (NW)	0
*R*8	*R*8	*R*8	Constant	1	$F = A + 1$	00001	Data In	Write (WR)	PC	Write (WR)	1
					$F = A + B$	00010					
					$F = A + B + 1$	00011					
					$F = A + \overline{B}$	00100					
					$F = A + \overline{B} + 1$	00101					
					$F = A - 1$	00110					
					$F = A$	00111					
					$F = A \wedge B$	01000					
					$F = A \vee B$	01010					
					$F = A \oplus B$	01100					
					$F = \overline{A}$	01110					
					$F = B$	10000					
					$F = \text{sr } B$	10100					
					$F = \text{sl } B$	11000					