Control Unit

010.133
Digital Computer Concept and Practice
Spring 2013

Lecture 06

Machine Instructions

- Machine instruction
  - A group of bits that specifies an operation and the registers or memory words in which the operands are found and the result is stored
  - Either all the same size or different sizes
- Operation code (opcode)
  - A group of bits in an instruction that specifies an operation
  - N-bit opcode can represent 2^n different operations
- The way how bits are organized in a machine instruction varies with the type of the instruction and the machine

Instruction Set

- Instruction set
  - A complete collection of instructions for a computer
- Instruction set architecture (ISA)
  - A thorough description of the instruction set
- Micro-architecture
  - The design techniques used to implement the instruction set

Data Transfer Instructions

- Load and store instructions that move data to and from memory and CPU registers
- Input and output instructions that moves data to and from CPU registers and I/O devices

Arithmetic, Logic, and Shift Instructions

- Addition, subtraction, multiply, and division instructions
- Bitwise AND, OR, and NOT instructions
- Logical and arithmetic shift instructions
- Comparison instructions that compare two values

Control Flow Instructions

- Unconditional branch instructions that jump to another location in the program to execute instructions there
- Conditional branch instructions that jump to another location in the program when a certain condition holds

Stored Program Concept

- The key idea of the von Neumann architecture
- Not only are all data values used in the program stored in memory, but also are machine instructions in the program
- Machine instructions are placed in adjacent locations and fetched by the CU one by one

Instruction Cycle

- Fetch-decode-execute cycle
  - Repeated until the computer is powered down
- Fetch
  - The CU fetches an instruction from memory
- Decode
  - The CU (instruction decoder) determines what operations the instruction requires
- Execute
  - The CU activates the necessary sequence of microoperations (i.e., control words) to provide timing and control signals to the datapath and memory
  - The decoder converts the instruction to control signals to the datapath and to the CU itself

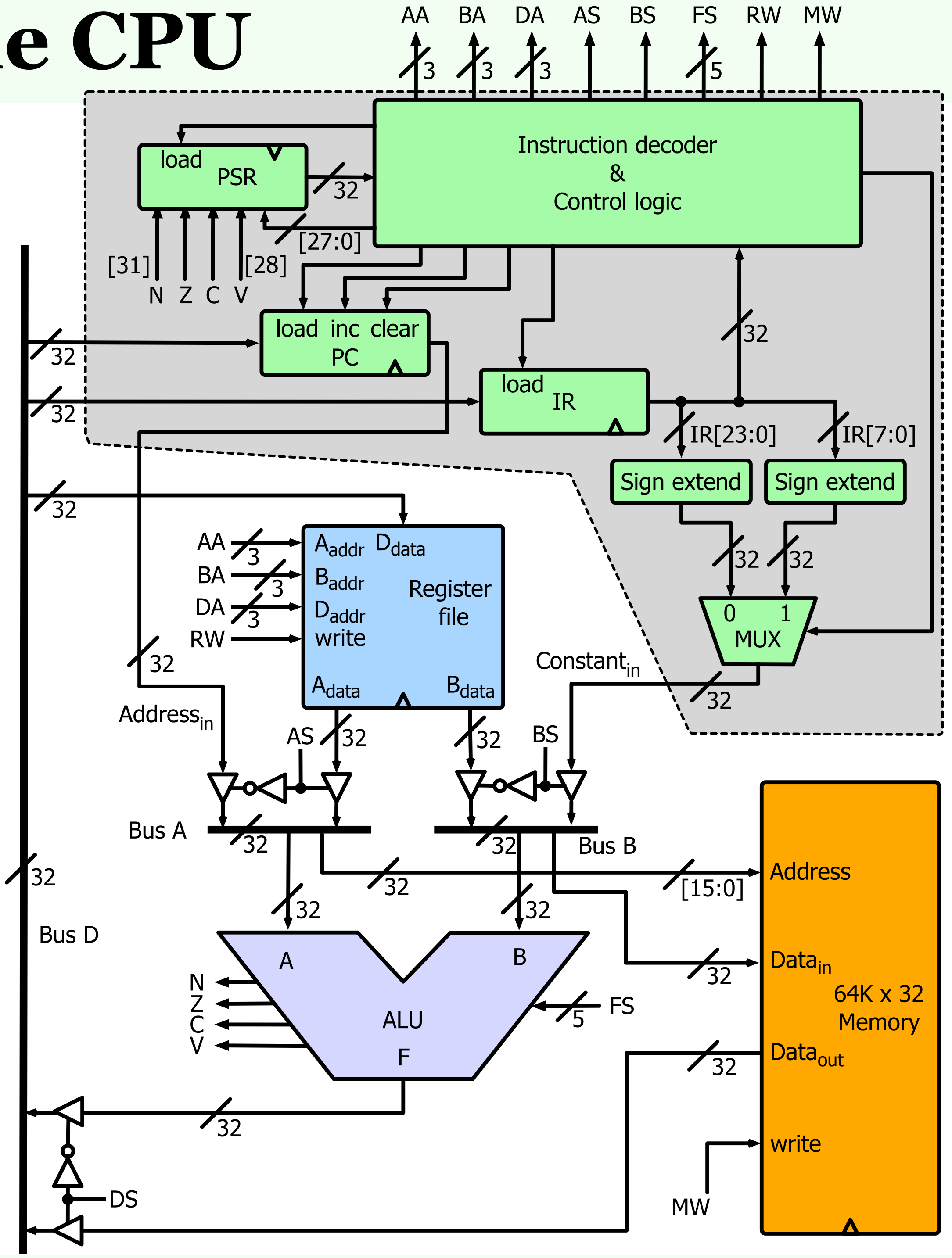
Program Counter

- A register to specify the address of the next instruction to be executed
  - To execute instructions in sequence
  - Either automatically incremented or loaded with a new address by the CU

Branch Instructions

- Modify the PC to skip over some sequence of instructions or to go back to repeat the previous instruction sequence
- Contain an offset
  - This offset is added to the current PC to go to the branch target address
- Conditional branches
  - Modify the PC when a certain condition is true
  - The CU evaluates the condition by checking the status signals from the datapath
- Unconditional branches
  - Always modify the PC
  - Always jumps to the branch target address

A Simple CPU


AA
BA
DA
AS
BS
FS
RW
MW
3
3
3
5
Instruction decoder & Control logic
load
PSR
32
[27:0]
[31]
[28]
N Z C V
load inc clear PC
load IR
32
IR[23:0]
IR[7:0]
Sign extend
Sign extend
32
32
0 1 MUX
Constant$_{in}$
32
AA
BA
DA
RW
A_{addr}
D_{data}
B_{addr}
D_{addr}
write
Register file
A_{data}
B_{data}
Address$_{in}$
AS
BS
Bus A
Bus B
Bus D
[15:0]
Address
Data$_{in}$
64K x 32 Memory
Data$_{out}$
write
MW
A
B
ALU
F
FS
5
DS

Instruction Register and Processor Status Register

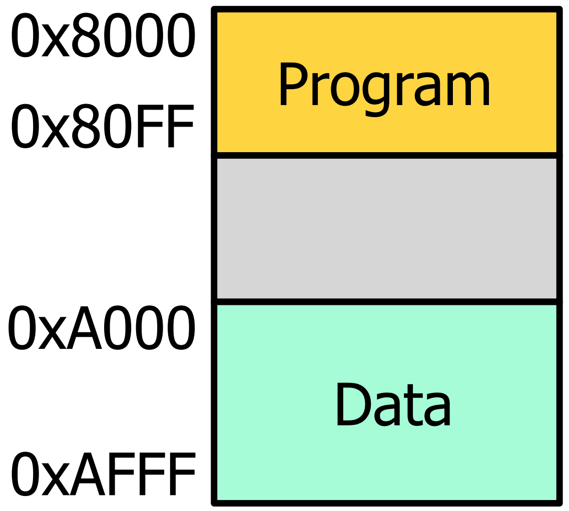
- The IR contains the current instruction fetched from memory
- The PSR is used by the CU to keep track of various aspects of the CPU state
- Status flags are set by a comparison instruction
  - PSR[31] ← N
  - PSR[30] ← Z
  - PSR[29] ← C
  - PSR[28] ← V

Status Flags

- N (negative)
  - The result of the last ALU operation is negative (MSB = 1)
- Z (zero)
  - The result of the last ALU operation is zero
- C (carry)
  - The result of the last ALU operation has a carry-out
- V (oVerflow)
  - The result of the last ALU operation overflows

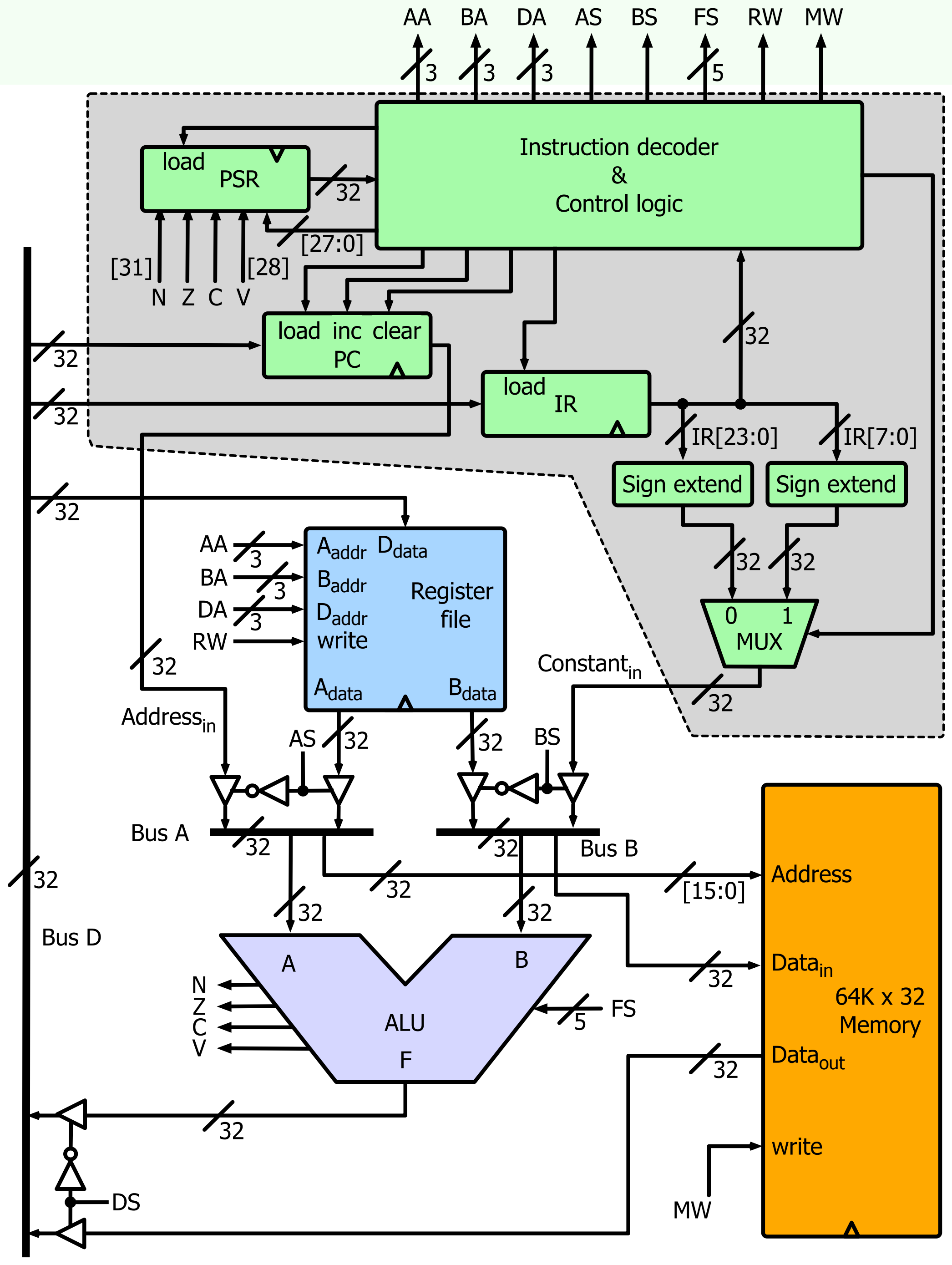
Memory Map

- Initially, the PC is loaded with 0x8000
- 0x8000 is the address of the first instruction to be executed
- The CPU repeats the fetch-decode-execute cycle



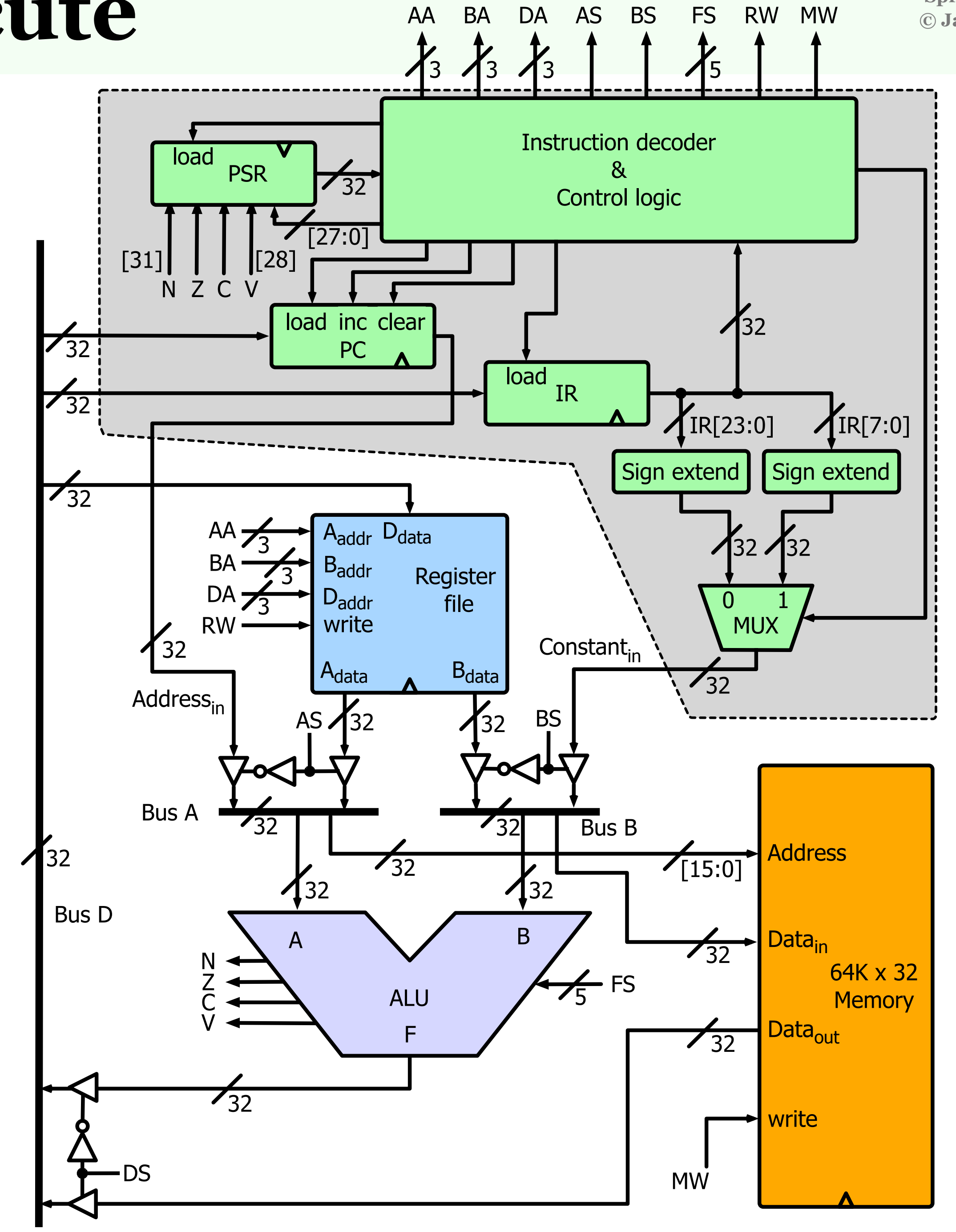
Fetch

- IR ← M[PC]


AA
BA
DA
AS
BS
FS
RW
MW
3
3
3
5
Instruction decoder & Control logic
load PSR
32
[27:0]
[31]
[28]
N Z C V
load inc clear PC
load IR
32
IR[23:0]
IR[7:0]
Sign extend
Sign extend
0 1 MUX
Constant_in
A_addr D_data B_addr D_addr write Register file A_data B_data
Address_in
AS
BS
Bus A
Bus B
Bus D
[15:0]
Address
Data_in
64K x 32 Memory
Data_out
write
MW
A B ALU F
FS
DS

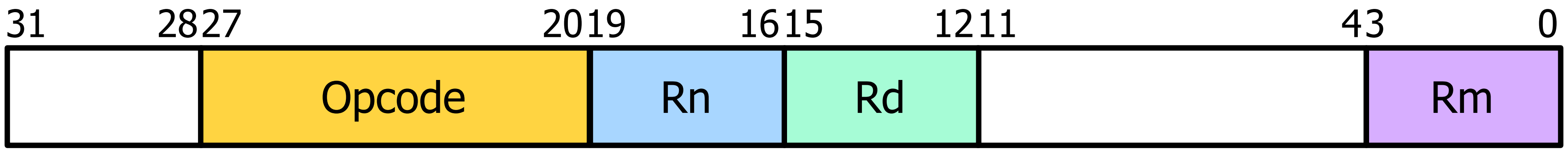
Decode and Execute

- The instruction decoder in the CU reads the content of the IR, and the opcode and operands are being decoded
- The CU generates appropriate control words to perform the operation specified by the opcode



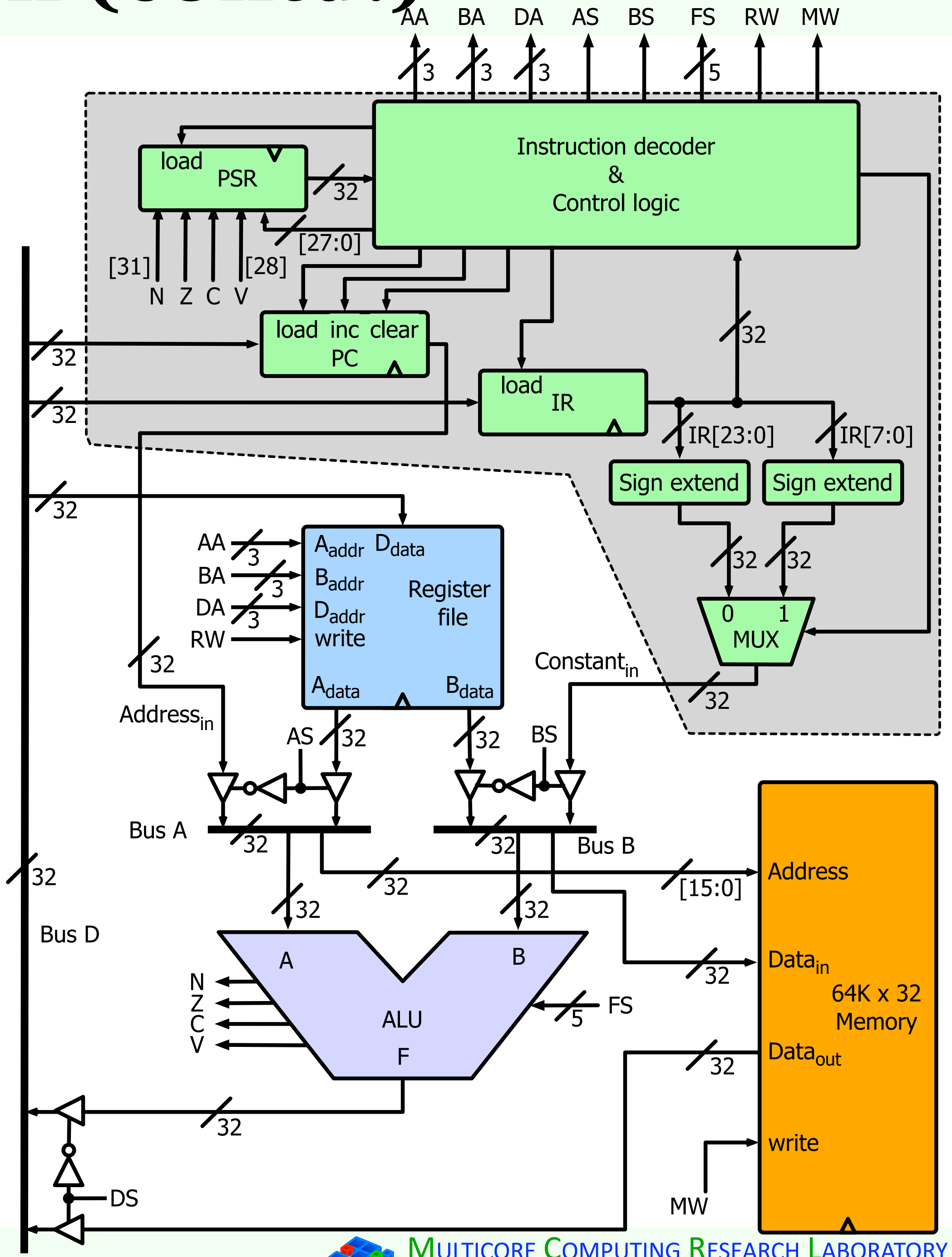
Addition Instruction

- An addition instruction that adds the contents of two registers R1 and R2 and stores the result to the register R3



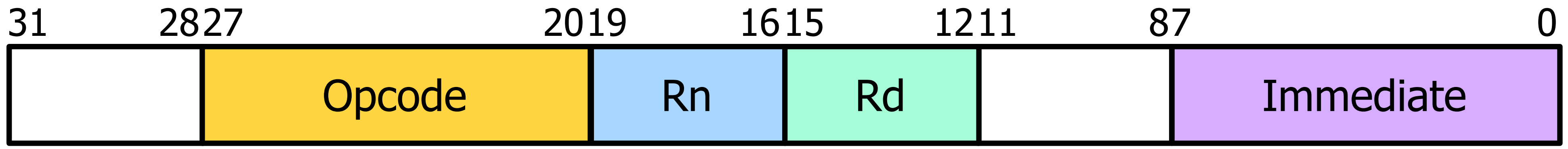
Addition Instruction (contd.)

- IR ← M[PC]
- R3 ← R1 + R2;
  PSR[31:28] ← NZCV;
  PC ← PC + 1
- Instruction cycle time
  - Total two clock cycles



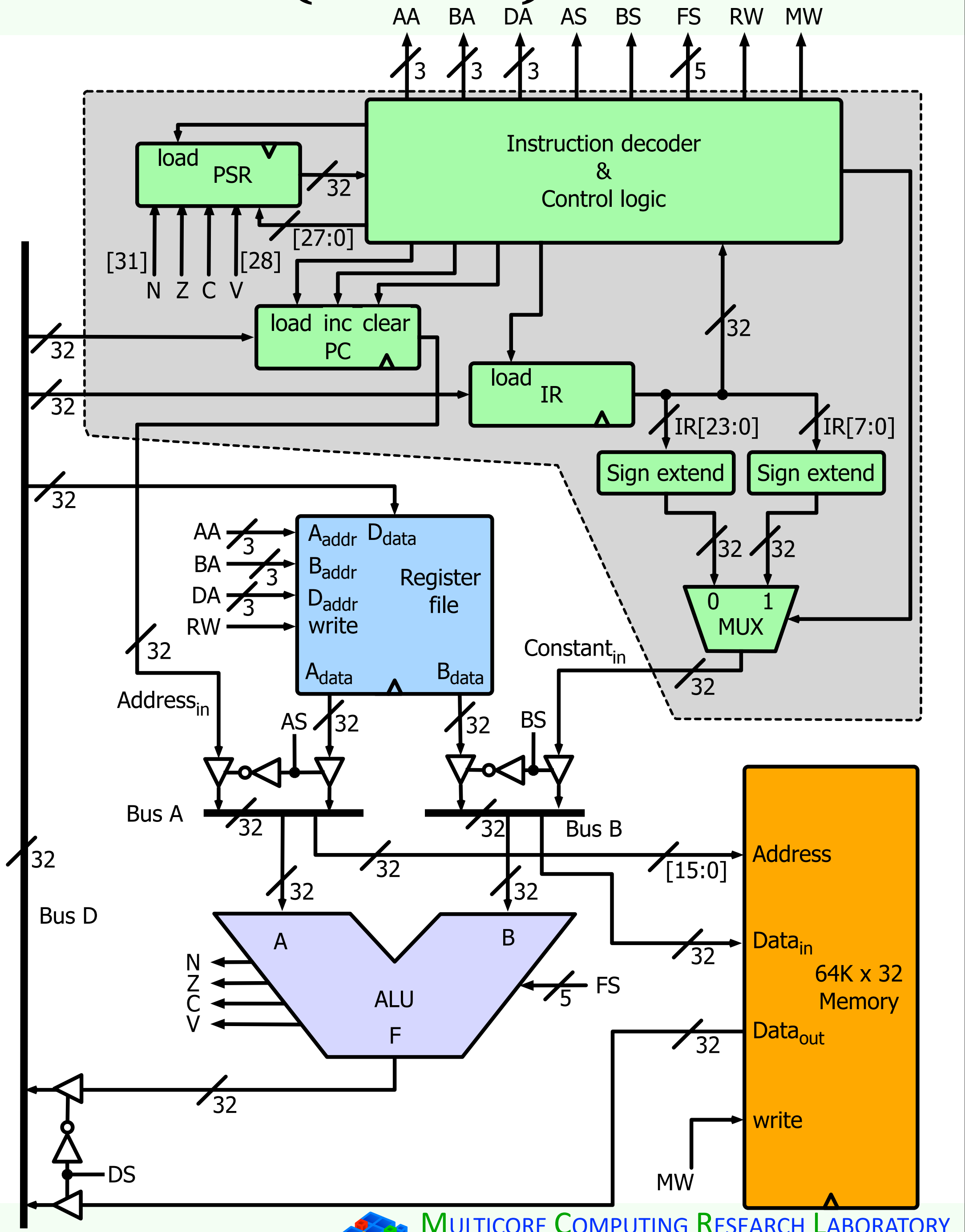
Immediate Addition Instruction

- An addition instruction that adds the content of register R1 and a constant 34, and stores the result to destination register R3
- 34
  - An immediate or an immediate constant
  - 8-bit signed binary number



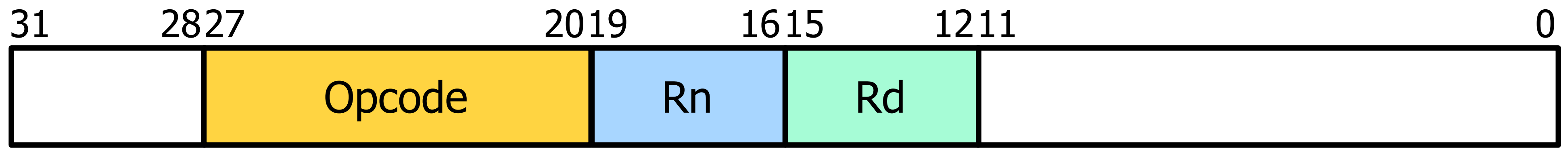
Immediate Addition Instruction (contd.)

- $IR \leftarrow M[PC]$
- $R3 \leftarrow R1 + Constant_{in}$;
  $PSR[31:28] \leftarrow NZCV$;
  $PC \leftarrow PC + 1$



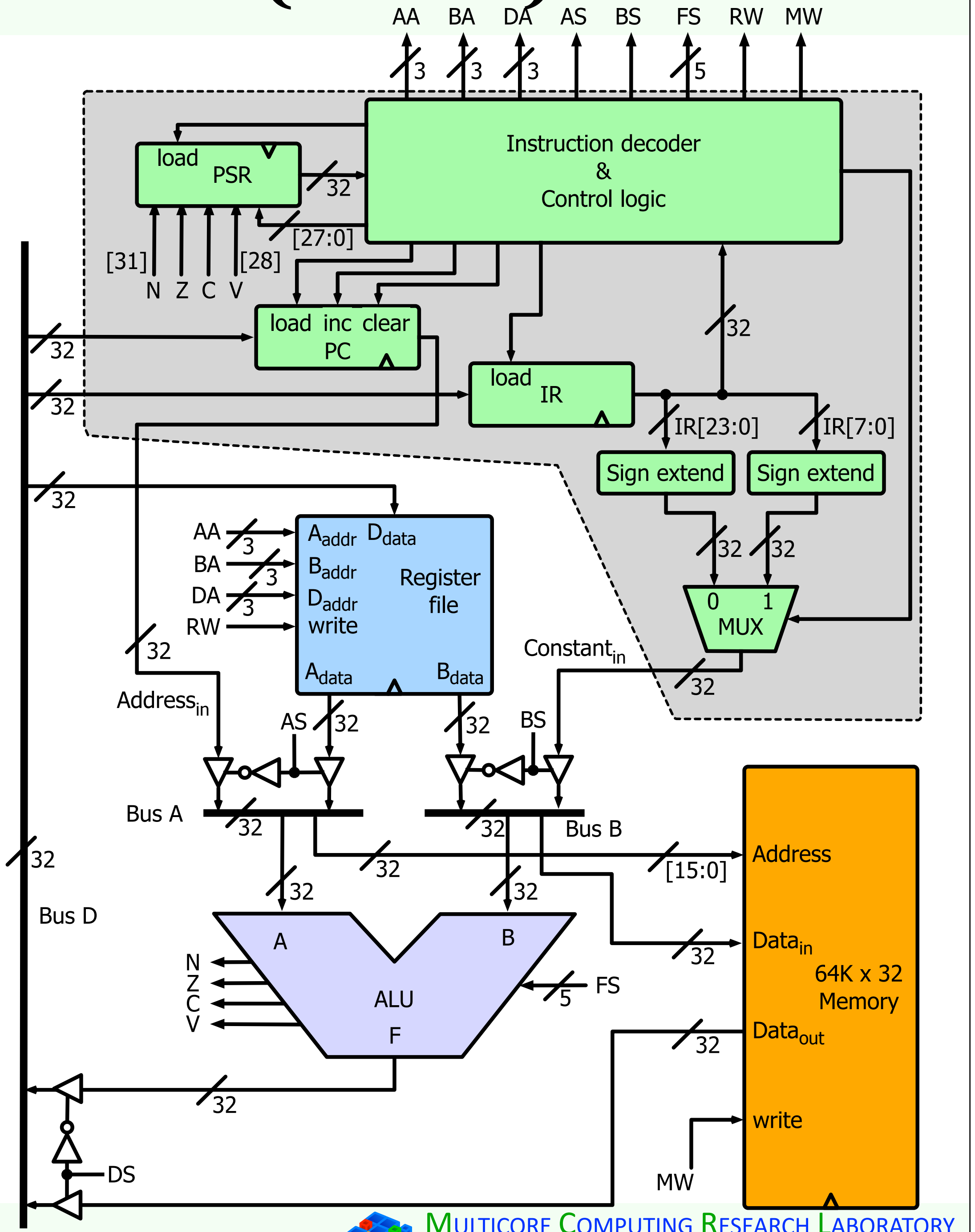
Load and Store Instructions

- Rn
  - Contains the address of the memory location
- Rd
  - The destination register when the instruction is a load instruction
  - The source register when the instruction is a store instruction



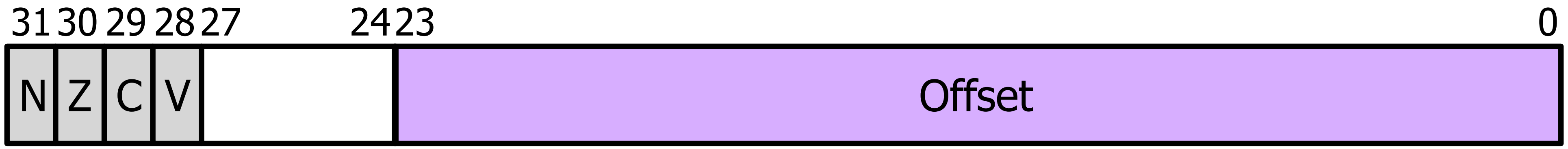
Load and Store Instructions (contd.)

- Load
  - IR ← M[PC]
  - Rd ← M[Rn];
    PC ← PC + 1
- Store
  - IR ← M[PC]
  - M[Rn] ← Rd;
    PC ← PC + 1



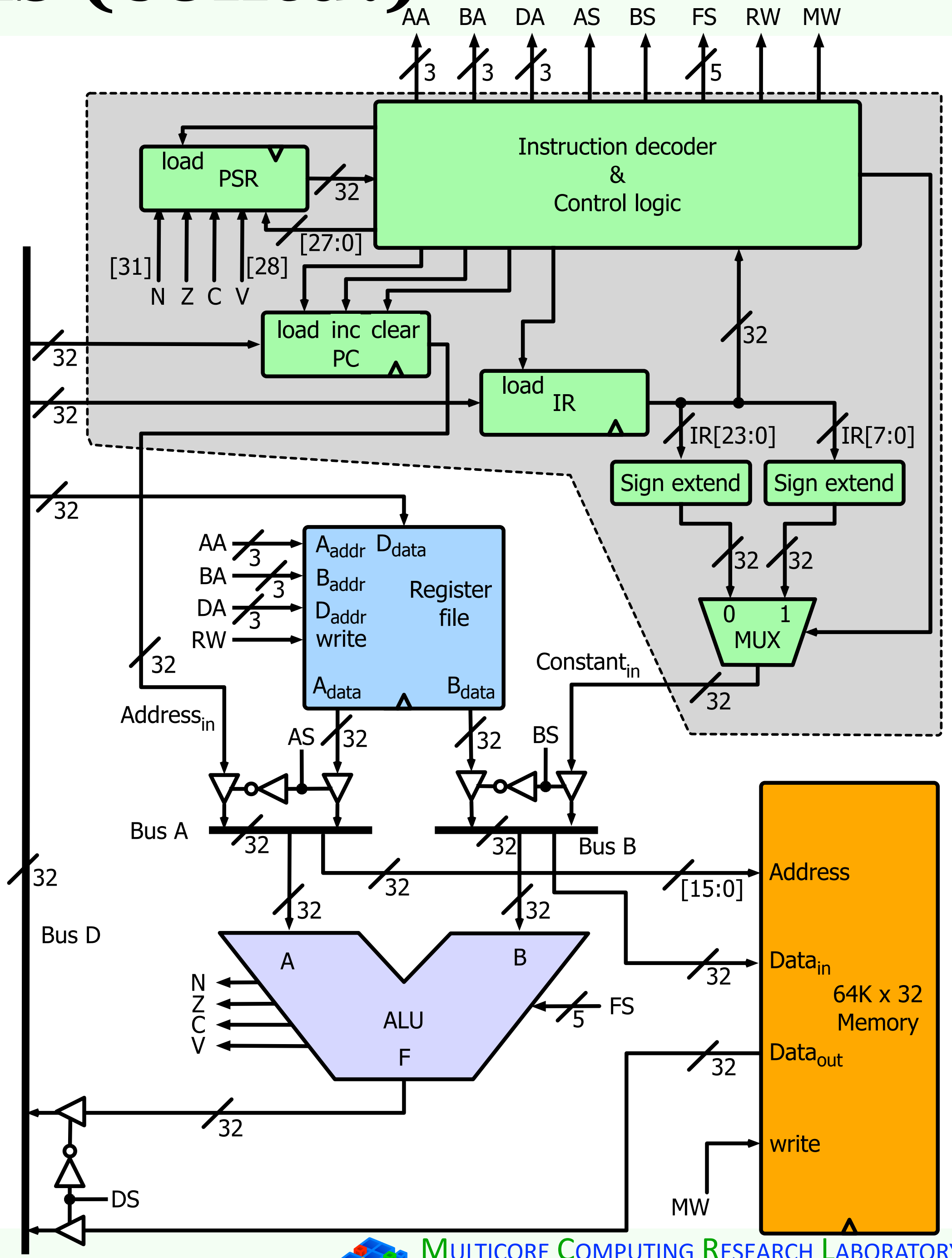
Branch Instructions

- Branch instruction alters the content of the PC
- 24-bit offset
  - Signed binary number in the two's complement representation
- Target address = the content of PC + offset



Branch Instructions (contd.)

- Unconditional branch
  - IR ← M[PC]
  - PC ← PC + $\text{Constant}_{\text{in}}$
- Conditional branch
  - IR ← M[PC]
  - Taken
    - PC ← PC + $\text{Constant}_{\text{in}}$
  - Not taken
    - PC ← PC + 1

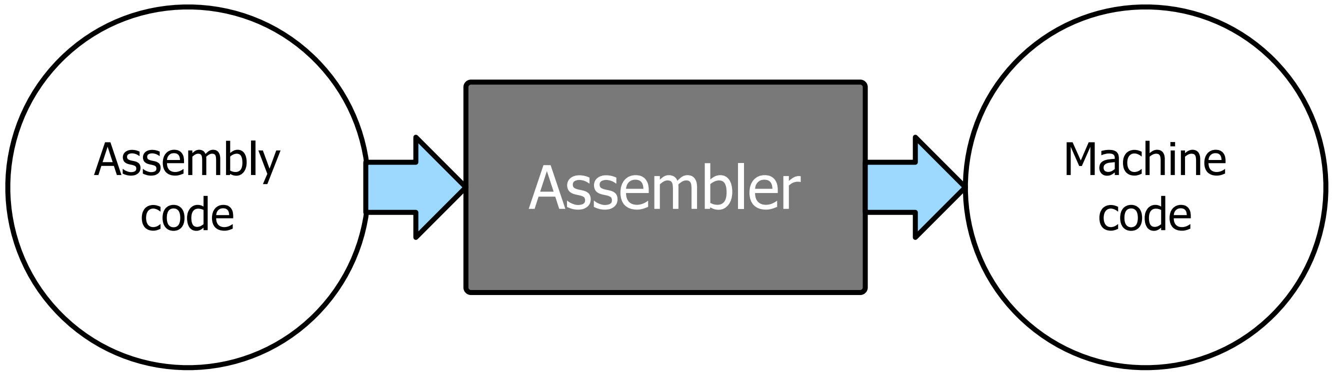


Assembly Language

- A low-level language and relatively easy to write a program compared to the machine language
    - Symbolic names for opcode (mnemonics), locations in the program (labels), variables, and constants
- Humans almost never write programs directly in machine code
    - Very difficult to understand and write a program in patterns of 0 and 1
    - Very much error prone

Assembly Language (contd.)

- An assembler directive is a command to the assembler that tells the assembler something to do in the assembly process
- A pseudo-instruction does not actually exist in the machine instruction set
    - An easy way of representing a group of machine instructions (possibly a single machine instruction)



Adding 10 Arbitrary Numbers Stored in Memory

```
MOV   R0, #0
MOV   R2, #99
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
ADD   R2, R2, #1
LDR   R1, [R2]
ADD   R0, R0, R1
```

```
     MOV   R0, #0
     MOV   R2, #99
L :  ADD   R2, R2, #1
     LDR   R1, [R2]
     ADD   R0, R0, R1
     CMP   R2, #109
     BNE   L
```

Microoperations for the Assembly Code

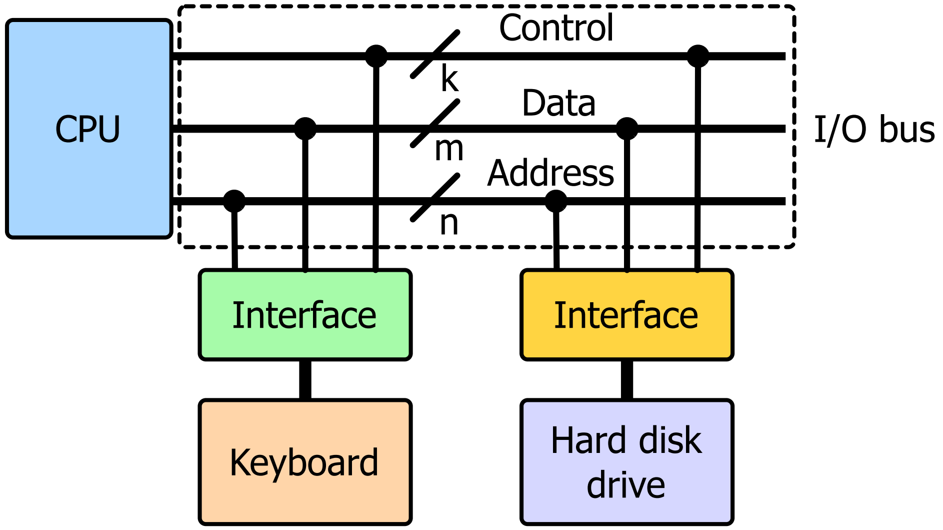
Instruction	T_0	T_1
MOV R0, #0	$IR \leftarrow M[PC]$	$R0 \leftarrow 0;$ $PC \leftarrow PC + 1$
MOV R2, #99	$IR \leftarrow M[PC]$	$R2 \leftarrow 99\ (Constant_{in});$ $PC \leftarrow PC + 1$
ADD R2, R2, #1	$IR \leftarrow M[PC]$	$R2 \leftarrow R2 + 1;$ $PC \leftarrow PC + 1$
LDR R1, [R2]	$IR \leftarrow M[PC]$	$R1 \leftarrow M[R2];$ $PC \leftarrow PC + 1$
ADD R0, R0, R1	$IR \leftarrow M[PC]$	$R0 \leftarrow R0 + R1;$ $PC \leftarrow PC + 1$
CMP R2, #109	$IR \leftarrow M[PC]$	$R0 \leftarrow R2 - 109\ (Constant_{in});$ $PSR[31:28] \leftarrow NZCV;$ $PC \leftarrow PC + 1$
BNE L	$IR \leftarrow M[PC]$	$PC \leftarrow PC + Constant_{in}$ if PSR[31] $= 0$ $PC \leftarrow PC + 1$ otherwise

Input and Output

- I/O devices attached to a computer is also called peripherals
  - Keyboards, mice, display units, speakers, printers, hard disk drives, optical disk drives, solid state disk drives, network interface cards, etc.
- Peripherals that communicate with people typically transfer alphanumeric information to/from the CPU
  - The standard binary code for the alphanumeric information is ASCII

I/O Bus

- An interface is required to resolve differences between the peripheral and CPU
  - Contains an address decoder, a control unit, and registers for the device
  - Has a distinct address
- To communicate with a specific peripheral device
  - The CPU places the address of the device on the address lines
  - Address lines are continuously monitored by the interface for each device
  - If the interface for a device detects its own address on the bus, a communication link is established
  - All other devices are disabled for the bus



Memory-mapped I/O

- Makes all I/O devices look exactly the same to the CPU
- Each I/O device is allocated to an exclusive portion of the CPU's address space
    - When a CPU has n-bit addresses, its address space is the set of 2^n possible addresses
- Normal load or store instructions are used to communicate with I/O devices
    - The instruction set of the CPU does not need to include special I/O instructions

Memory-mapped I/O (contd.)

- To enable memory-mapped I/O,
  - Each I/O device needs to provide a hardware interface similar to that of memory
  - Each I/O device is required to define an interaction contract (protocol)
- The exclusive portion of the address space allocated to an I/O device continuously reflects the physical state of the device
  - Pressing a key on the keyboard makes a certain value (e.g., ASCII code of the key) to be written in the area allocated to the keyboard
  - Whenever a bit is changed in the area allocated to a physical screen, the associated pixel is drawn on the screen