Chapter 04: Instruction Sets and the Processor organizations

Lesson 10:

Processor Instructions - Part 3

Objective

- Learn Instructions
- Shift and Rotate
- Multiplication and division
- Input and Output

Shift

Arithmetic and logic Left shifts ASL and LSL

- **Arithmetic shift left— ASL** Shift the bits to left side positions at the first input operand by the number of positions specified in the second input operand (or a default implied number).
- **logical shift left — LSL** Shift left like ASL [ASL and LSL gives same result]

Arithmetic shift right ASR

- **Arithmetic shift Right ─ ASR** Shift the bits to right side positions at the first input operand by the number of positions specified in the second input operand (or a default implied number) and shift as well retain the old sign bit (msb)

Logic shift right LSR

- **Logic shift Right – LSR** Shift the bits to right side positions at the first input operand by the number of positions specified in the second input operand (or a default implied number)
- ASR and LSR are distinct operations

Example Instruction LSL 25, 2

- Assume: A system with 8-bit data words
- The 8-bit integer representation of 25 is 0b 0001 1001
- Shifting this to the left two bit positions gives 0b011001vv
- "v" bits indicate vacant bits

Post shift the Filling of vacated Positions

- A LSH left operation specifies that zeroes be shifted into vacant bit positions, so the final result is 0b01100100, which is the 8-bit binary representation of 100

Instruction LSR 25

- Default second input operand will be assumed 1
- The 8-bit integer representation of 25 is 0b00011001
- Shifting this to the right one bit position gives 0bv0001100, and lsb bit 1 is shifted out
- "v" bits indicate vacant bits

Post shift the Filling of vacated Positions

- A LSH right operation specifies that zeroes be shifted into vacant bit positions
- Result is 0b00001100, which is the 8-bit binary representation of 12

LSR – 16, 2

- The 8-bit two's-complement representation of –16 is 0b1111 0000
- Using LSR to shift right 2 bits (a shift of –2 positions), gives 0b001111 00, the two's-complement representation of +60 because shifting a 0 into the high bit position of a two's-complement integer makes the result a positive value
- ASR operation— provides the solution for signed integer cases

ASR –16, 2

- 0b1111 0000 will become 0bvv111100 and v will be 1 as msb = 1
- Therefore result is 0b11111100 the two's-complement representation of –4
- There is division of –16 by 2^2

8-bit two's-complement representation +16 Multiply it by 5

- When we use the instruction. we get 0b010 00000. Now ADD the result with the original number itself
- Therefore, result is 0b010 00000 + 0b0001 0000 = 0b0101 0000 the representation of +60
- There is multiplication by $2^2 + 1$

Rotate

Rotate Left and Right RL and RR

- **Rotate left RL—** Rotate the bits (lsb to left side positions and msb to vacant positions) at the first input operand clockwise
- **Rotate Right — RR** Rotate the bits (msb to right side position and lsb to vacant positions) at the first input operand anticlockwise

Rotate left by instruction RL 144

- The 8-bit unsigned 144 binary bits = 0b1001 0000
- We get 0b0010 000v and v will be from the msb, therefore the result is 0b0001 0001 and Carry C = v-bit = 1

Rotate right by instruction RR 65

- 8-bit unsigned 65 binary bits = 0b0100 0001
- When we use the instruction RR 65, we get 0bv010 0000 and v will be from the lsb
- Result is 0b1010 0000 and Carry C = v-bit = 1

Multiplication and division

Multiplication and division Instructions

- Multiplication and division— Complex arithmetic operations provided in modern processors
- Take one or two inputs, and generate one output
- In general, arithmetic operations read their inputs from and write their outputs to the register file

MUL and FMUL

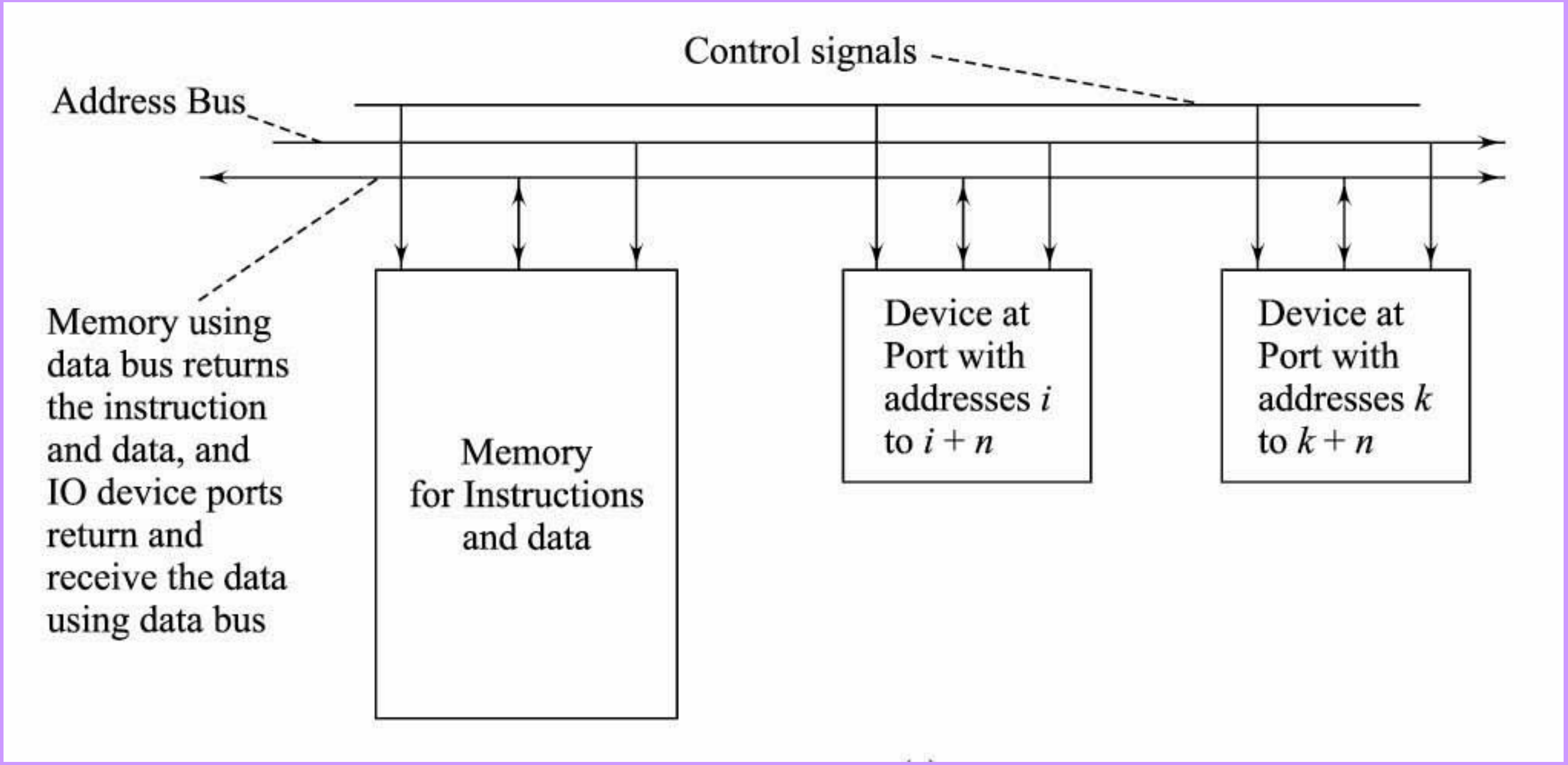
- **Multiply– MUL** Multiply its second integer operand to its first
- **Floating Point Multiply – FMUL** Multiply its second floating point operand to its first

DIV, FDIV and NEG

- **Divide— DIV** Divide its second integer operand from its first
- **Floating Point Divide — FDIV** Divide its second floating point operand from its first
- **Negate— NEG** Negate the input operand, 23 will be – 23 and – 423 will be + 423

Input and Output

Input and output Instructions when separate control signals for IOs



Input and Output

- **Input — IN** Load accumulator with bits from a device port, the address of which is the operand in the instruction
- **OUT— OUT** Store accumulator bits to a device port, the address of which is the operand in the instruction

Summary

We learnt

- **Instructions** Shift and Rotate
- Multiplication and division
- Input and Output

End of Lesson 10 on

Processor Instructions - Part 3