# Lab3

2022年1月20日 14:21

4:0 9:5 21:10

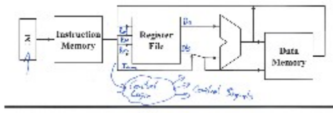
31 22

**Instruction set:** 101000 1000

- ✓ ADDI Rd, Rn, Imm12: Reg[Rd] = Reg[Rn] + ZeroExtend(Imm12).
- ✓ ADDS Rd, Rn, Rm: Reg[Rd] = Reg[Rn] + Reg[Rm]. Set flags.
- ✓ AND Rd, Rn, Rm: Reg[Rd] = Reg[Rn] & Reg[Rm].
- ✓ B Imm26: PC = PC + SignExtend(Imm26 << 2).
  For lab #4 (only) this instr. has a delay slot.
- ✓ B.LT Imm19: If (flags.negative != flags.overflow) PC = PC + SignExtend(Imm19<<2).
  For lab #4 (only) this instr. has a delay slot.
- ✓ CBZ Rd, Imm19: If (Reg[Rd] == 0) PC = PC + SignExtend(Imm19<<2). [illegible]
  For lab #4 (only) this instr. has a delay slot.
- ✓ EOR Rd, Rn, Rm: Reg[Rd] = Reg[Rn] ^ Reg[Rm].
- ✓ LDUR Rd, [Rn, #Imm9]: Reg[Rd] = Mem[Reg[Rn] + SignExtend(Imm9)].
  For lab #4 (only) the value in rd cannot be used in the next cycle.
- ✓ LSR Rd, Rn, Shamt: Reg[Rd] = Reg[Rn] >> Shamt [5:0]
- ✓ STUR Rd, [Rn, #Imm9]: Mem[Reg[Rn] + SignExtend(Imm9)] = Reg[Rd].
- ✓ SUBS Rd, Rn, Rm: Reg[Rd] = Reg[Rn] - Reg[Rm]. Set flags.

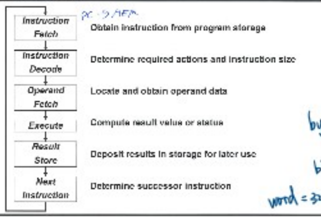
## Processor Overview

Overall Dataflow
- PC fetches instructions
- Instructions select operand registers, ALU immediate values
- ALU computes values
- Load/Store addresses computed in ALU
- Result goes to register file or Data memory



73

## Execution Cycle



Instruction Fetch — Obtain instruction from program storage
Instruction Decode — Determine required actions and instruction size
Operand Fetch — Locate and obtain operand data
Execute — Compute result value or status
Result Store — Deposit results in storage for later use
Next Instruction — Determine successor instruction

byte 大

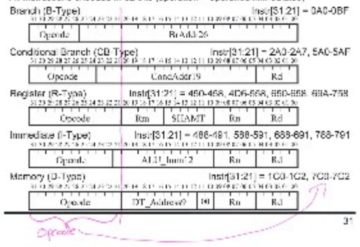
bit 小

word = 32 bit = 4byte

72

## Instruction Formats

All instructions encoded in 32 bits (operation + operands/immediates)

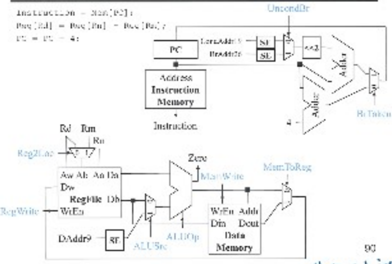


Branch (B-Type) Instr[31:21] = 0A0-0BF
Opcode | BrAddr26
Conditional Branch (CB Type) Instr[31:21] = 2A0-2A7, 5A0-5AF
Opcode | CondAddr19 | Rd
Register (R-Type) Instr[31:21] = 450-458, 4D6-558, 650-658, 69A-758
Opcode | Rm | SHAMT | Rn | Rd
Immediate (I-Type) Instr[31:21] = 488-491, 588-591, 688-691, 788-791
Opcode | ALU_Imm12 | Rn | Rd
Memory (D-Type) Instr[31:21] = 1C0-1C2, 7C0-7C2
Opcode | DT_Address9 | 00 | Rn | Rd
Opcode

31

## SUB Control

```
Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
```



90

4byte = 1 instru

8bit

Instru memo size 1024 byte. 1024/4 = 256 instrutions

input address 64bits — byte's address

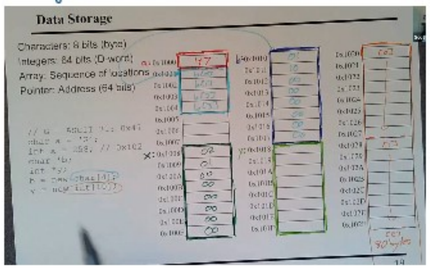
output instruction 32 bits

4x8 = 32.

4 byte

or → hex hex

memory:

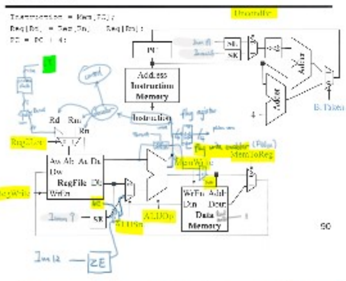


**Data Storage**

Characters: 8 bits (byte)
Integers: 64 bits (D-word)
Array: Sequence of locations
Pointer: Address (64 bits)

## SUB Control

```
Instruction = Mem[PC];
Reg[Rn] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
```



90

BrTaken = 🟩 (AL#0F) & (flags.negative != flags.overflow) | UncondBr | CBZ & zero

from flag register — control — control — immediate flag ×from register

✓ controller: input: 11 bit instruction [31:21]
output: controls (in highlight)

✓ 3 SE: input 9bit, 19 bit, 26bit,
output ↓ 64bit, ↓ 32bit, ↓ 32bit

✓ 1 ZE: input: 12 bit
output: ↓ 64bit

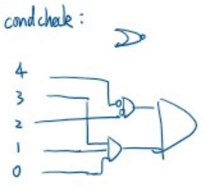
✓ shifter: input 64 bit **in Math.SV**
output 64 bit

✓ 32bit_adder: input two 32bit
output one 32bit

✓ flag_register: input flags [3:0], writeEnabler
use 4 D-ff — 3 negative, 2 zero, 1 of, 0 carryout

output: flag_out [3:0]

✓ condcheck:



## Control Signals

| Opcode[31:26] / Opcode[25:21] | 100100 0100x | 101010 11000 | 111110 00010 | 111110 00000 | 000101 xxxxx | 101101 00xxx | 010101 00xxx | 100010 10000 | 110010 10000 | 110100 11010 | 111010 11000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDI | ADDS | LDUR | STUR | B | CBZ | B.cond (LT) 01011 | AND | EOR | LSR | SUBS |
| Reg2LOC | X | 1 | X | 0 | X | 0 | X | 1 | 1 | X | 1 |
| AdI | 1 | X | 0 | 0 | X | X | X | X | X | X | X |
| ALUSrc | 1 | 0 | 1 | 1 | X | 0 | X | 0 | 0 | X | 0 |
| 2:0 ALUOP | add 010 | add 010 | add 010 | add 010 | X | Bypass B 000 | X | AND 100 | XOR 110 | X | SUB 011 |
| FWen | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Sctr | 0 | 0 | 0 | 0 | X | X | X | 0 | 0 | 1 | 0 |
| MemToReg | 0 | 0 | 1 | X | X | X | X | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| instru Bcond | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| instru UncondBr | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| instru CBZ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | |

**Instruction set:**

ADDI Rd, Rn, Imm12: Reg[Rd] = Reg[Rn] + ZeroExtend(Imm12).
ADDS Rd, Rn, Rm: Reg[Rd] = Reg[Rn] + Reg[Rm]. Set flags.
AND Rd, Rn, Rm: Reg[Rd] = Reg[Rn] & Reg[Rm]. 11 [31:21] 450
B Imm26: PC = PC + SignExtend(Imm26 << 2). 6 [31:26] B
For lab #4 (only) this instr. has a delay slot.
B.LT Imm19: If (flags.negative != flags.overflow) PC = PC + SignExtend(Imm19<<2). 8 [31:24] 0B
For lab #4 (only) this instr. has a delay slot.
CBZ Rd, Imm19: If (Reg[Rd] == 0) PC = PC + SignExtend(Imm19<<2). 8 [31:24] B4
For lab #4 (only) this instr. has a delay slot.
EOR Rd, Rn, Rm: Reg[Rd] = Reg[Rn] ^ Reg[Rm]. 11 [31:21] 650
LDUR Rd, [Rn, #Imm9]: Reg[Rd] = Mem[Reg[Rn] + SignExtend(Imm9)]. 11 [31:21] 7C2
For lab #4 (only) the value in rd cannot be used in the next cycle.
LSR Rd, Rn, Shamt: Reg[Rd] = Reg[Rn] >> Shamt 11 [31:21] 69A
STUR Rd, [Rn, #Imm9]: Mem[Reg[Rn] + SignExtend(Imm9)] = Reg[Rd]. 11 [31:21] 7C0
SUBS Rd, Rn, Rm: Reg[Rd] = Reg[Rn] - Reg[Rm]. Set flags. 11 [31:21] 758