单周期处理器的微体系结构
ARM7TDMI 简化

yann.douze@sorbonne-universite.fr

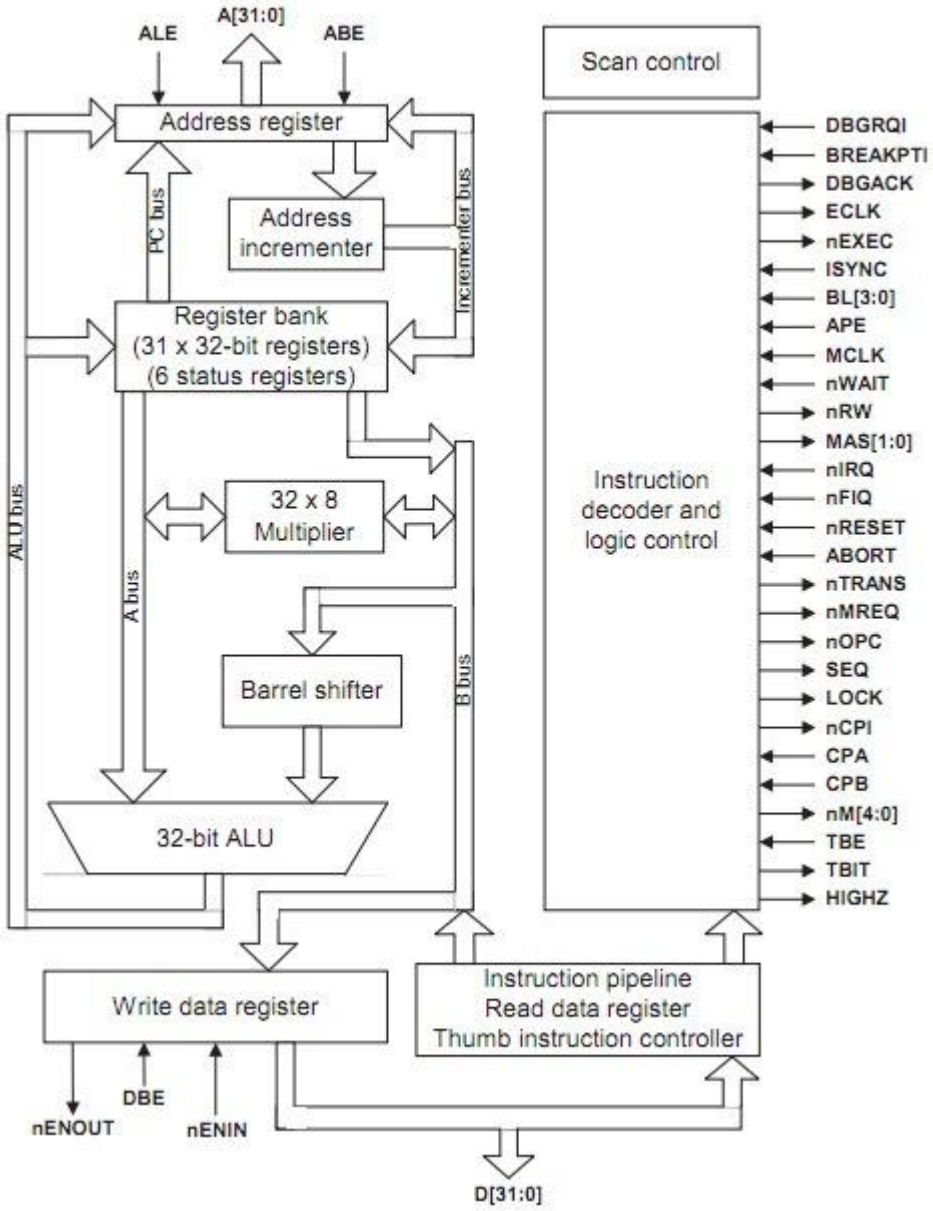
介绍

- 课程目标：描述单周期处理器的架构
- 以 ARM7TDMI 处理器指令集为例
- 第 1 部分：ARM7TDMI 处理器架构介绍
- 第 2 部分：如何逐步设计处理器

第1部分：ARM7TDMI架构

- 32-bit处理器
- 32-bit ALU
- 32x8寄存器队列
- 移位寄存器■

32x8 乘法器



主要特点

- 加载- 存储架构
  - 指令仅处理寄存器中的数据并将结果放入寄存器中。唯一的那些访问内存的操作是将内存值复制到寄存器（加载）的操作和将寄存器值复制到内存（存储）的操作。
- 指令的固定编码格式
  - 所有指令均以32 位编码。
- 处理指令的3种地址格式
  - 2 个操作数寄存器和1 个结果寄存器，可独立指定。
- 条件执行
  - 每条指令都可以有条件地执行
- ALU+移位
  - 可以在一条指令（1个周期）中执行算术或逻辑运算和移位，它们在大多数其他处理器上由单独的指令执行的。

处理单元

- UT功能单元:
  - 寄存器文件
  - 16 个寄存器，用于灵活处理数据和存储 ALU 结果
- 算术和逻辑单元。

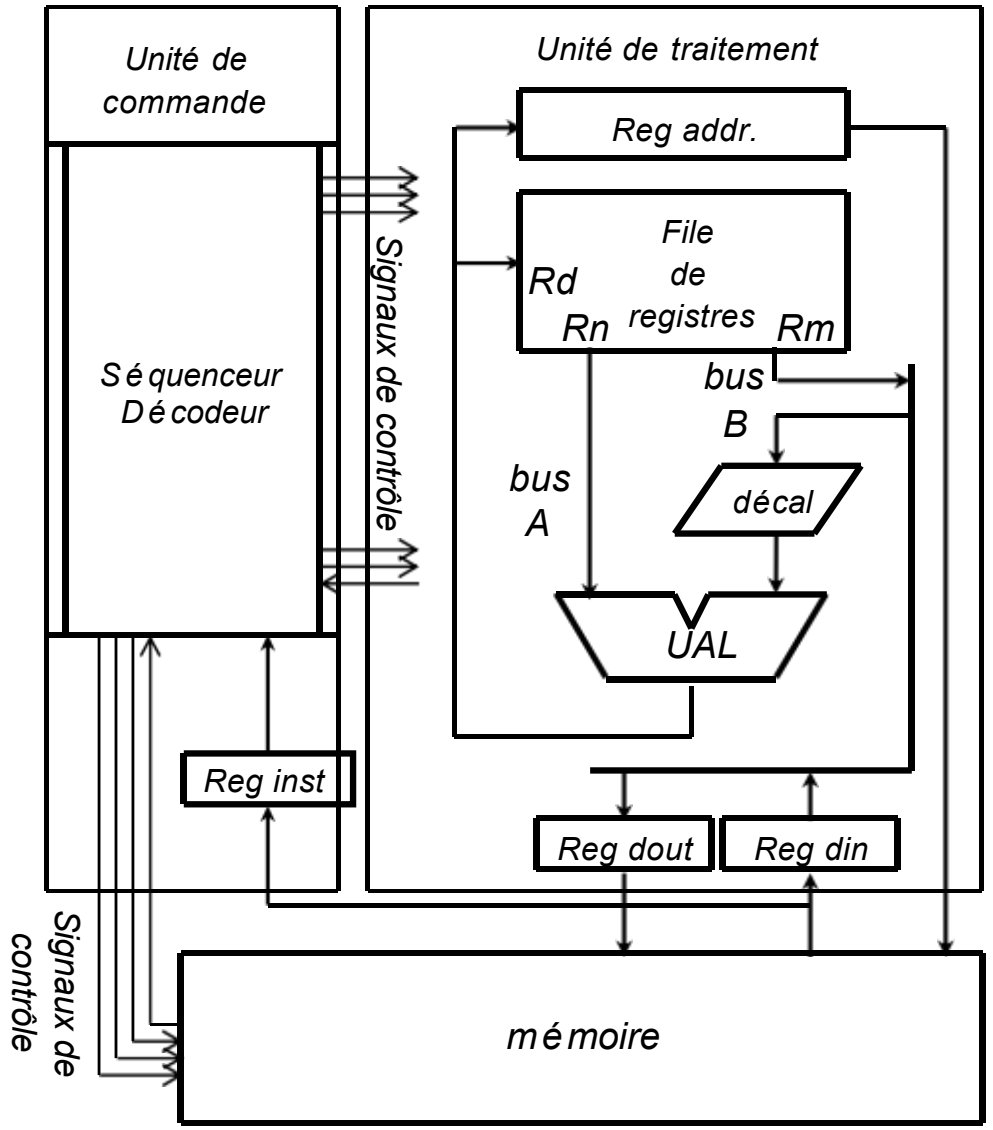
  2个操作数:来自寄存器的输入A数据，连接到移位器的输入 B数据

  ALU 结果:返回寄存器
- 移位寄存器

  移位操作(左移1次=x2，右移1次= 12)

  循环操作(移位+重新注入丢失的比特)

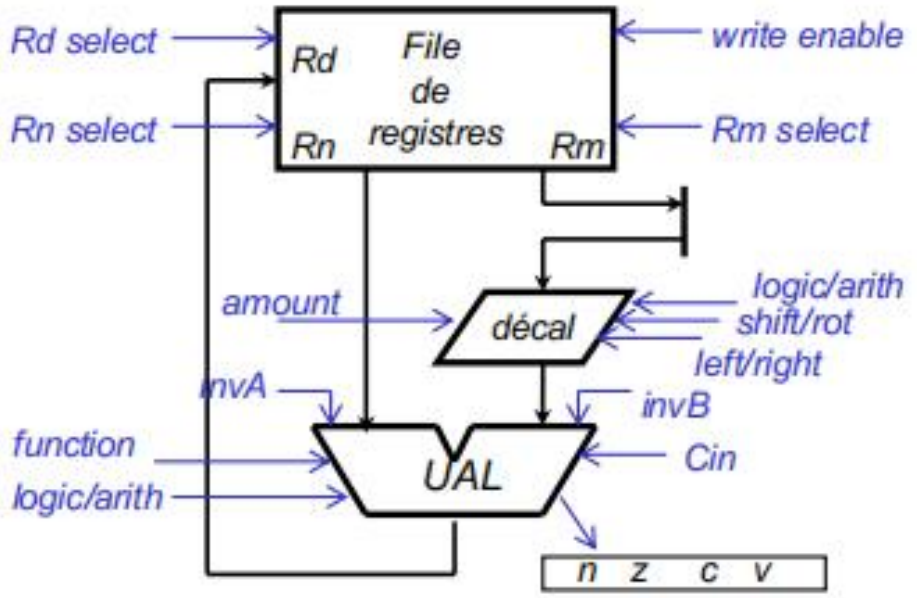
  与 ALU 输入B相连，在一个周期内执行 ALU+移位指令



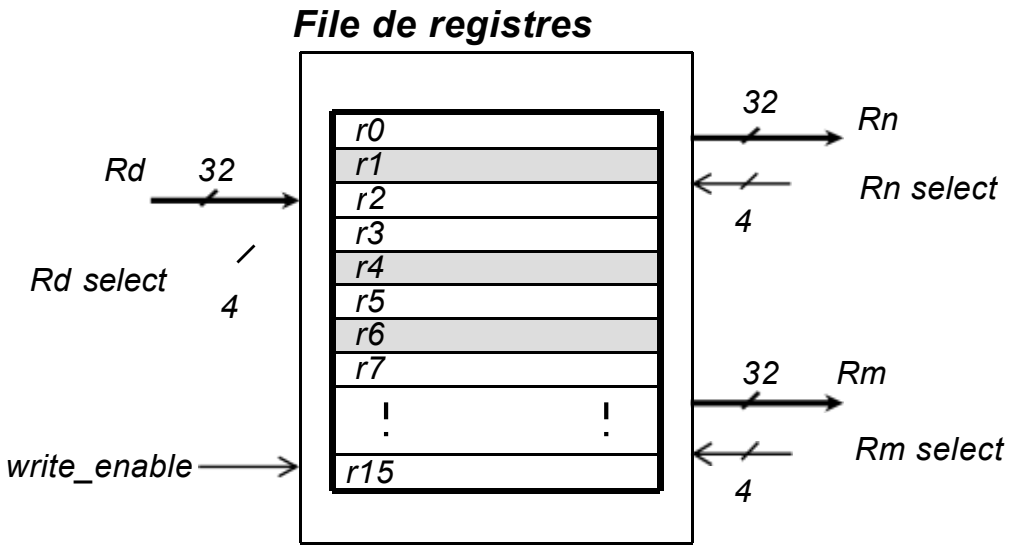
处理单元结构

- 处理单元的控制信号

  - 寄存器文件
    - Rd, Rn, Rm select：
      在16个寄存器队列中选择目 标寄存器
    - write enable:激活寄存器队列
  - ALU
    logic/arith+function ：逻辑或算术模式，在每种模式下可选择功能
    InvA、invB:操作数A和B的反转(非)
    Cin:注入保持位C
    指示器:指示 ALU 运算(如保持 C)后的状态
  - Décaleur
    - shift/rot:移位或循环操作
    - logic/arith:逻辑或算术运算
    - Amount:移位/循环位数



寄存器文件File de registres



- 16 用户寄存器 r0, …, r15
- 调用指令的符号:
  - INST Rd, Rn, Rm
    - 3地址格式 (2个操作数+结果)
    - Rn 寄存器操作数 1，连接到端口 A(32 位)
    - Rm 寄存器操作数 2，通过移位器连接到端口 B(32 位)-->输入B可以进行旋转/
    - Rd 目的地寄存器(32 位)
    - 选择 4 位，从 16 个可用寄存器中选择一个

MOV指令

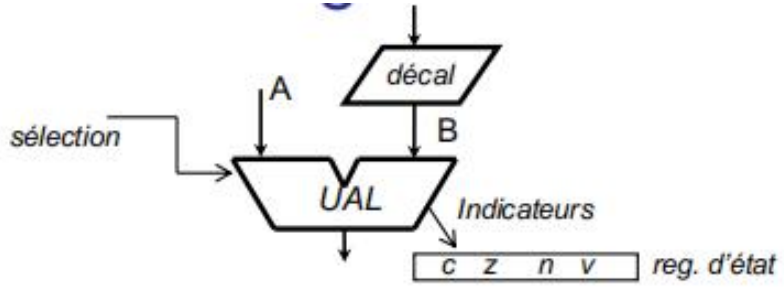
- 寄存器之间的数据移动指令
  - MOV (Move), MVN (Move not)
    - MOV Rd, *#literal*
    - MOV Rd, Rn
    - MOV Rd, Rm, *shift*

    - 寄存器之间的数据移动，或仅从常量到寄存器的数据移动。

    - *#literal*: 立即数(常量)
    - *shift*:第二个操作数可能需要移位

  - Exemples
    - `MOV r3, #2` `@ r3 ← 2`
    - `MOV r3, r4` `@ r3 ← r4`
    - `MOV r3, r4, LSL #2` `@ r3 ← r4<<2`

ALU 和状态寄存器



- 算术和逻辑运算（执行算术和逻辑运算）
  - (ADD, SUB, AND, OR, …)
- 状态寄存器 (SR) 提供运算结果的指示：
  - C: 进位
    - 算术(或移位)运算的溢出指示位
  - Z: Zero
    - UAL 零结果指示位
  - N:负
    - UAL负结果的指示位
  - V: 溢出(oVerflow)
    - 结果溢出的指示位(修改符号位)

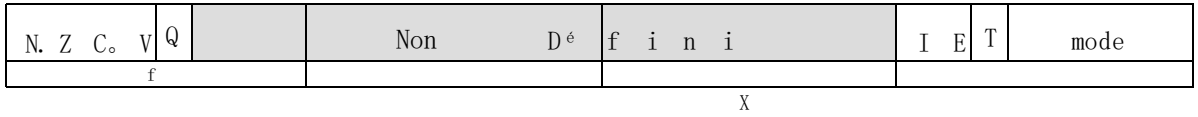
状态寄存器

- **4位示例**: 1 0 1 0 + 1 0 0 1 = (1) 0 0 1 1
  - ALU结果: 0 0 1 1
  - C = 1
  - Z = 0, 结果与 0 0 0 0 不同
  - N = 0, 0 0 1 1 是正数，因为符号位 (第 4 bit) = 0
  - V = 1, 因为 1 0 1 0 和 1 0 0 1 是负数，结果是正数

  - 根据对数字的解释，C、V、N 位将被检查或忽略
    - 如果数字采用无符号表示，则C有用，V和N无用
    - 如果数字是有符号表示，C没用，V和N有用
    - 这些标志位由UAL设置，程序员在使用或不使用它们时取决于其需要(例如，在示例的情况下从4位加法执行8位加法)

Le registre d'état:CPSR

- CPSR:当前程序状态寄存器
- 包含指示符Z（零）、N（负）、C（进位）、V（溢出）
- 给出算术运算或比较结果的信息
- 允许根据结果执行或不执行指令(条件执行)
- 允许为超过 32 位的操作保留进位



条件语句

- 在ARM上，可以有条件的执行所有指令：指令是否执行取决于位 N、Z、C、V 的状态
  - 在ARM汇编器中，只需在指令关键字上添加一个后缀，即可表示其执行条件

扩展助记符	解释	指示符状态
EQ	Equal/equals zero	Z = 1
NE	Not Equal	Z = 0
LT	**Lesser Than**	**N = 1**
GE	Greater than or equal	N = 0
VS	Overflow	V = 1
VC	Not overflow	V = 0

条件语句

- 基于比较结果的条件执行示例

```
CMP r4, r5                  @ comparer r4 et r5
SUBGT r4, r4, r5            @ si >, alors r4 ← r4 - r5
SUBLE r5, r5, r4            @ si ≤, alors r5 ← r5 - r4
```

- 基于计算结果的条件执行示例

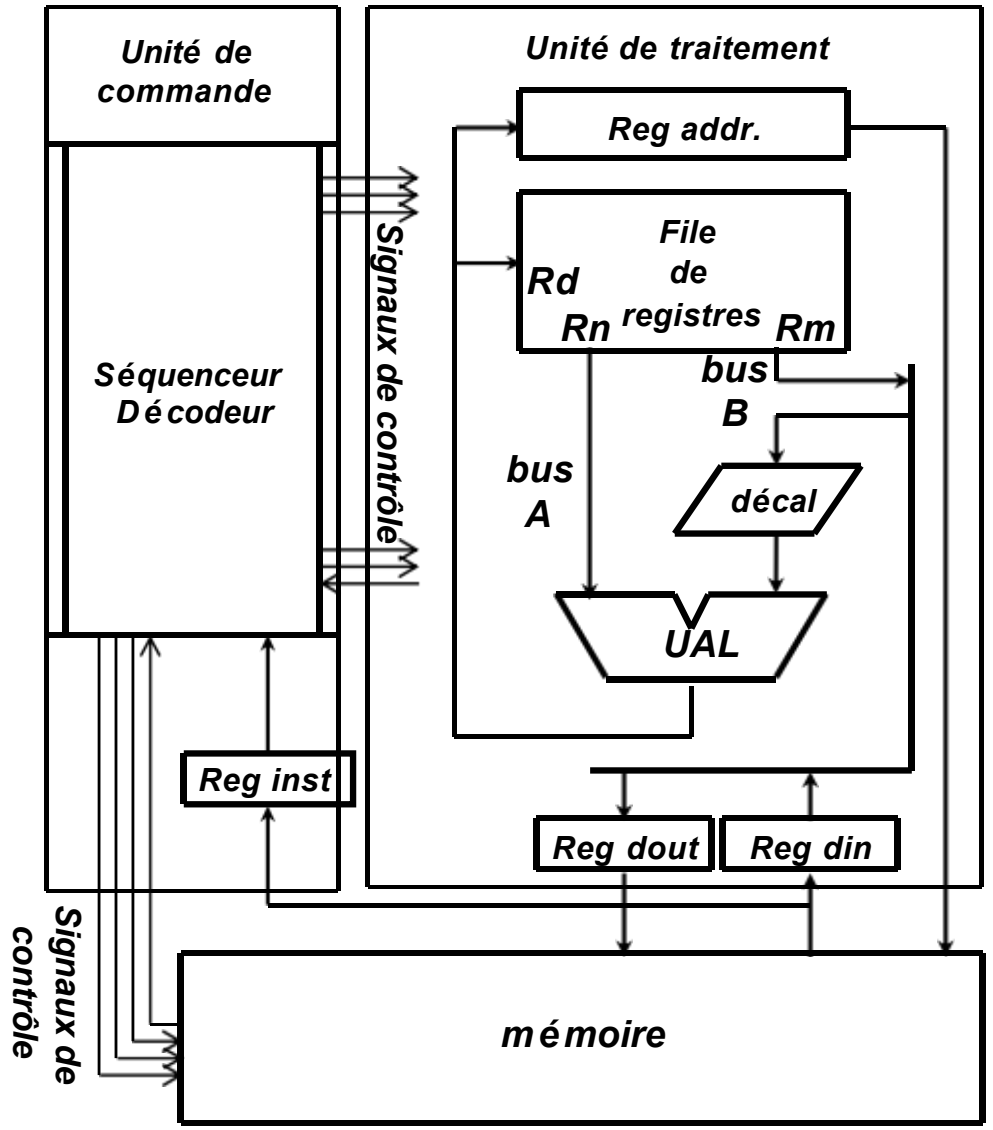
```
SUBS r4, r4, #10            @ r4 ← r4 - 10
MOVPL r5, #1                @ si r4>0, r5 ← +1
MOVMI r5, #-1               @ si r4<0, r5 ← -1
MOVEQ r5, #0                @ si r4=0, r5 ← 0
```

- 条件分支示例

```
ADDS r6, r4, r5             @ r6 ← r4 + r5
BLVS ErrDebordement               @ Branch and Link (BL) si VS
                            @ si débordement (VS),
                            @ appel du sous-programme
                            @ ErrDebordement
```

控制单元的作用

- 内存访问控制
  - 获取指令/数据
  - 设置访问内存中指令/数据所需的信号
- 解码指令
  - 解释说明
  - 将指令转化为控制信号的过程
- 处理单元控制
  - 执行指令
  - 设置执行指令所需的信号



第2部分：
如何逐步设计处理器

1. 解析指令集
2. 选择一组组件来创建数据路径，并建立时钟方法
3. 组装组件以创建数据路径
4. 建立控制逻辑

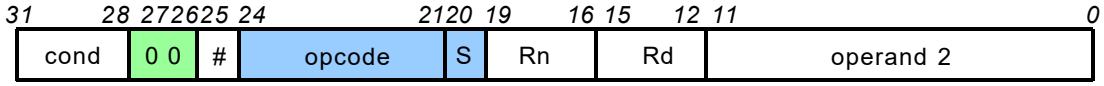
第1步：ARM7TDMI指令

- 数据处理：
  - ADD Rd, Rn, RM → Rd = Rn + Rm (Addition)
  - ADD Rd, Rn, #Imm → Rd = Rn + Imm (Addition)
  - MOV Rd, #Imm → Rd = Imm (Move)
  - CMP Rn, #Imm → Flag CPSR = Rn – Imm (Compare)
- 存储器访问：
  - LDR Rd, [Rn, #Offset] → Rd = Mem[Rn + Offset] (Load, lecture)
  - STR Rd, [Rn, #Offset] → Mem[Rn + Offset] := Rd (Store, écriture)
- 分支：
  - B{AL} label → PC = PC + Offset (Branch always)
  - BLT label → Si Flag N =1 => PC = PC + Offset (Branch if Lesser Than)

指令的二进制编码

- 处理指令由字段组成
  - 结果寄存器（目的地）
    - **Rd**: R0…R15
  - 第一个操作数（始终是寄存器）
    - **Rn**: R0…R15
  - 第二个操作数
    - **Rm**: R0…R15
    - Rm: R0…r15 带移位的(log/arith/rot, quantité)
    - 立即数 (常量)
  - 由ALU执行的操作
    - 操作码

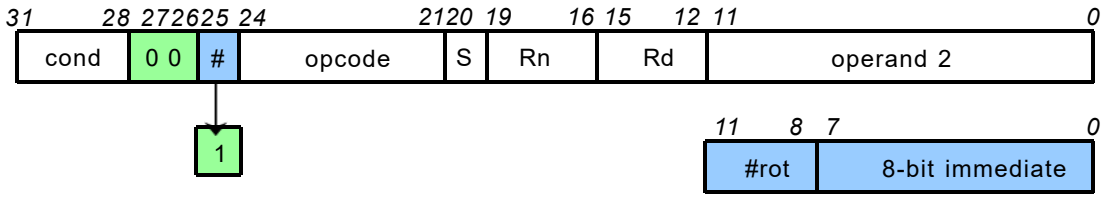
ARM7TDMI的数据处理指令



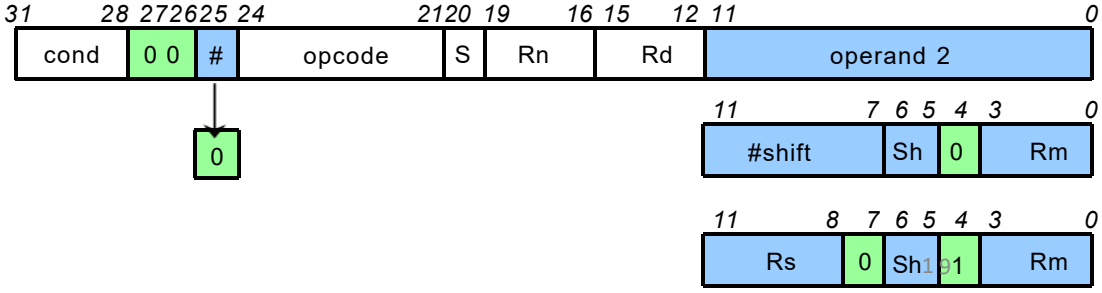
Asm	Opcode
AND	0000
EOR	0001
SUB	0010
RSB	0011
ADD	0100
ADC	0101

- *Rd* → 目标寄存器号
  - 4 位用于在 16 个可能的寄存器中进行选择
- *Rn* → 作为第一个操作数的寄存器号
  - 4 位用于在 16 个可能的寄存器中进行选择
- *operand2* → Rm 或 立即数
- 例如：ADD Rd, Rn, operand2 → Rd = Rn + operand2

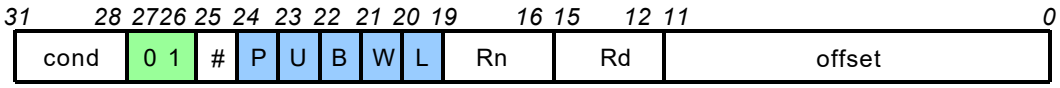
ARM7TDMI的数据处理指令



- 如果位# 为 1，则操作数 2为立即数
  - 示例：ADD R0, R1, #10 → R0 = R1 + 10
- 如果位#为 0，则操作数 2是一个寄存器(Rm)，被应用（或不应用）循环/移位
  - 示例：ADD R0, R2, R3 → R0 = R2 + R3



数据传输指令 (Load 加载 和 Store 存储)



- *B*：字节/字
  - 1：8位访问，0：32位访问
- *L*：加载/存储

  1 load, 0 store
- 示例：
  - LDR Rd, [Rn, #Offset] → Rd = Mem[Rn + Offset] (Load, 读取)
  - STR Rd, [Rn, #Offset] → Mem[Rn + Offset] := Rd (Store, 写入)

Load 加载 ——把数据从外部存储器读取到寄存器中
Store 存储——把数据从寄存器存储到外部存储器中

分支指令

- 分支 (B)

- 偏移：24 bits有符号移位
- Cond：条件执行
- BLT → 如果小于分支跳转
- 如果 CPSR的标志N = 1，则跳转

Asm	Cond
VS	0110
VC	0111
HI	1000
LS	1001
GE	1010
LT	1011

获取指令 (Fetch)

- 首先要检索指令
- PC：程序计数器

op | rn | rd | rm = MEM[PC]

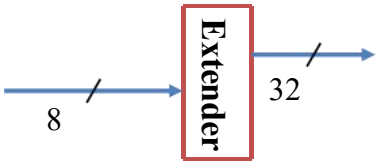
op | rn | rd | Imm8 = MEM[PC]

实例	寄存器传输	
ADD	**R[rd] <– R[rn] + R[rm];**	**PC <– PC + 1**
SUB	**R[rd] <– R[rn] – R[rm];**	**PC <– PC + 1**
LOAD	**R[rd] <– MEM[R[rn] + sign_ext(Imm8)];**	**PC <– PC + 1**
STORE	**MEM[R[rn] + sign_ext(Imm8)] <– R[rd];**	**PC <– PC + 1**

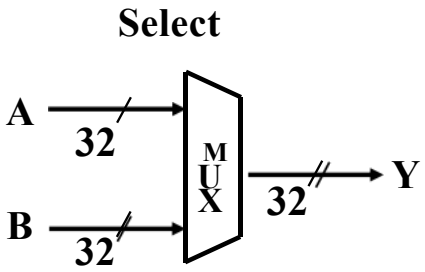
第2步：数据路径的元素
(datapath数据路径)

组合逻辑Logique Combinatoire

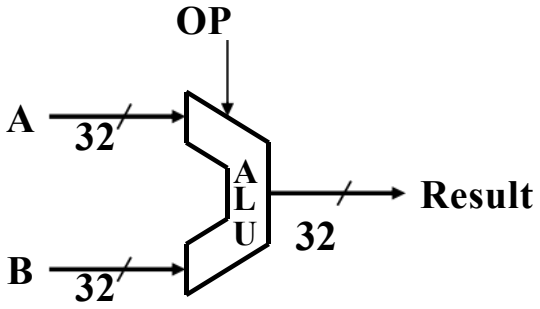
- 扩展器
Extender



- 数据选择器
MUX



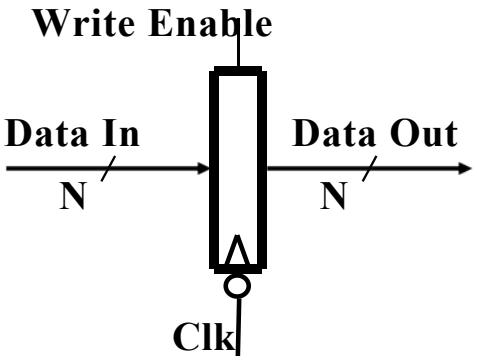
- 逻辑运算单元
ALU



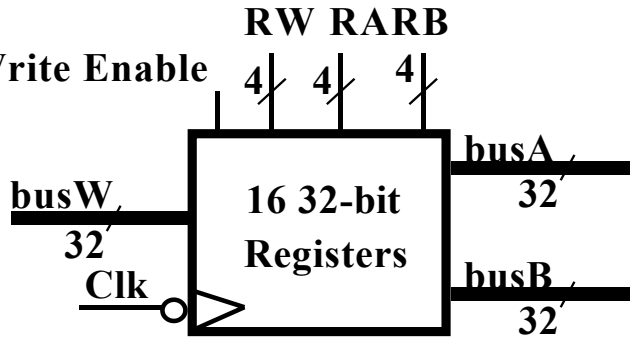
Carry
Zero
Sign
Overflow

存储原件：寄存器

- 寄存器
  - 类似于 D Flip-Flop
    - N-bit 输入与输出
    - 写使能
  - 写使能信号:
    - (0): 不修改输出数据
    - (1): Data Out 复制 Data In

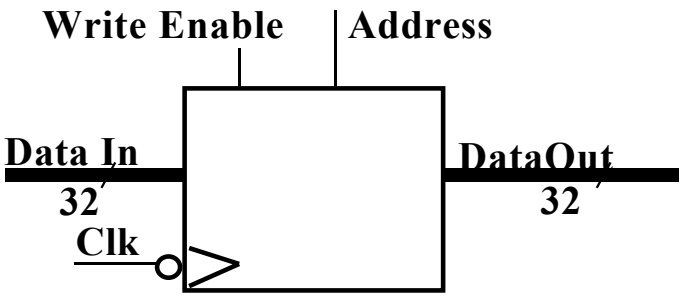


存储元件：寄存器组



- 寄存器组包含16 个寄存器：
  - 两个32位总线输出: busA 和 busB
    - 32位输入总线: busW
- 选择寄存器：
  - RA (编号)选择要放在总线A（数据）上的寄存器。
  - RB (编号)选择要放在总线B（数据）上的寄存器。
  - RW (编号)选择当写使能 = 1 时要从 W 总线写入哪个寄存器
- 时钟输入（CLK）
  - CLK 输入仅在写操作期间评估
  - 在读取操作过程中，其行为类似于组合逻辑块：
  - RA 或 RB 有效 => 访问时间后总线 A 或总线 B 有效。 （Tp：传播时间）

存储元件: 理想存储器

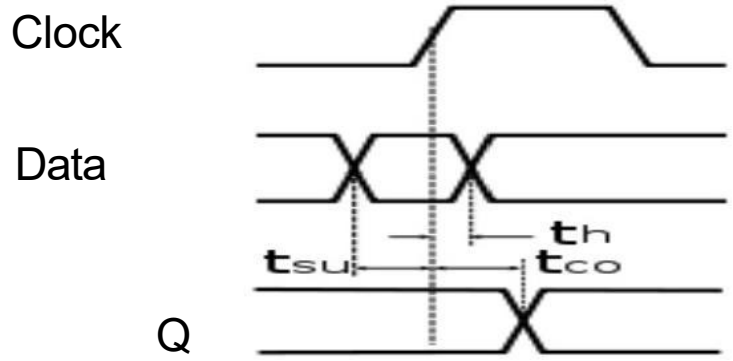


- 存储器（理想化）
  - 输入总线: Data In
  - 输出总线： Data Out

- 存储器字通过以下方式选择：
  - 地址：选择要输出数据的字
  - 写使能 = 1：地址选择要通过数据输入总线写入的存储器字

- 时钟输入（CLK）
  - CLK 输入仅在写操作期间评估。
  - 在读取操作期间，其行为类似于组合逻辑块：
  - 有效地址 => 访问时间 (Tp) 后有效数据输出

D 触发器的时序



·Tsu: 建立时间

——预设值时间

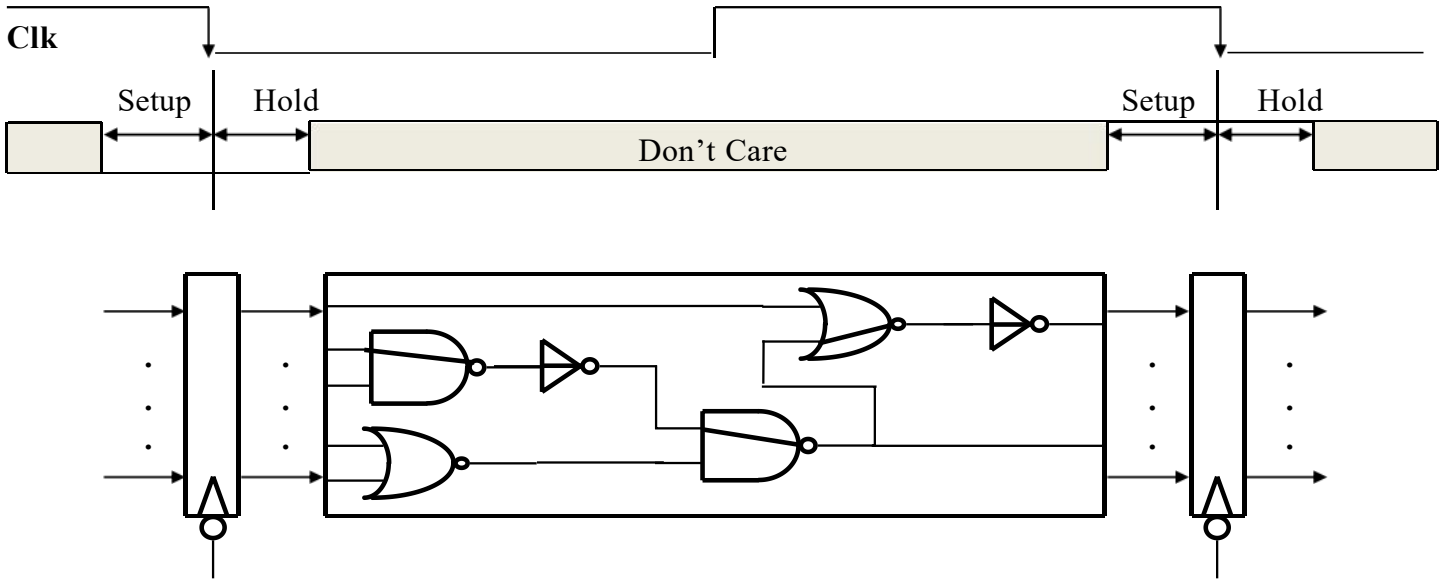
· Th: 保持时间

——保持一段时间

Tco: 从时钟端到输出端的时间Clock

Tp: 传播时间

RTL：寄存器传输级别



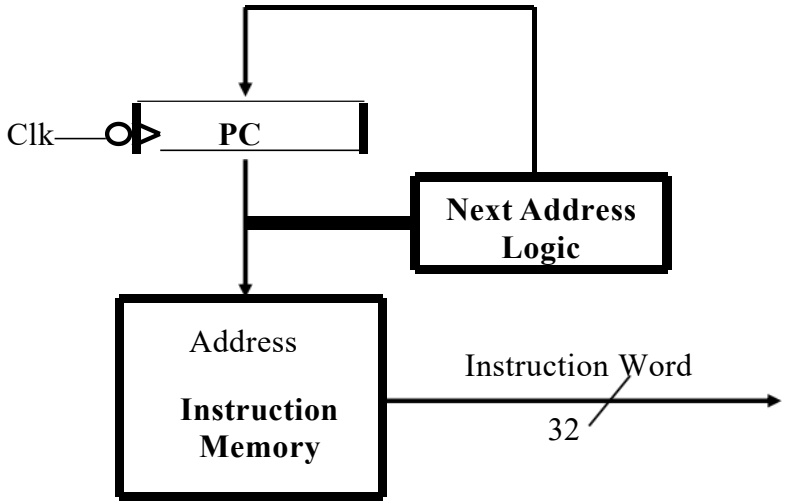
- 所有存储元件均由同一时钟沿提供时钟
- 周期时间 = CLK-to-Q (Tco) + 最长延迟路径 (Tp) + 建立时间 (Tsu)
- 处理器的最大工作频率 Fmax = 1/ 周期时间

步骤 3：组装组件以创建数据路径

- 检索指令(Fetch)
- 执行指令 -> 组装数据路径

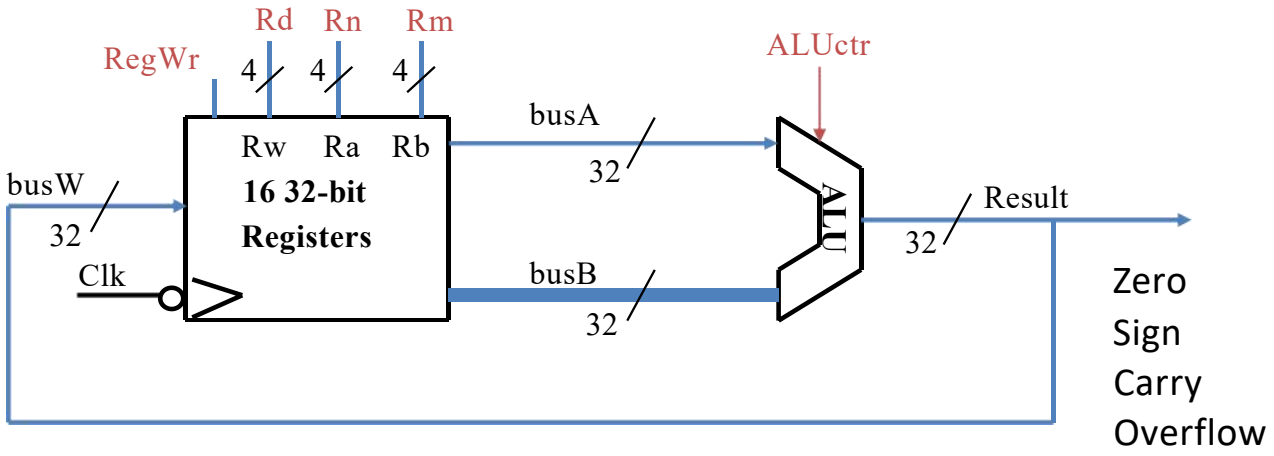
3a: 取指单元 (Fetch)

- 客观的
  - 获取指令: mem [PC]
  - 更新程序计数器:
    - 顺序代码：PC <- PC + 1
    - 分 支 ： PC <-“其他”

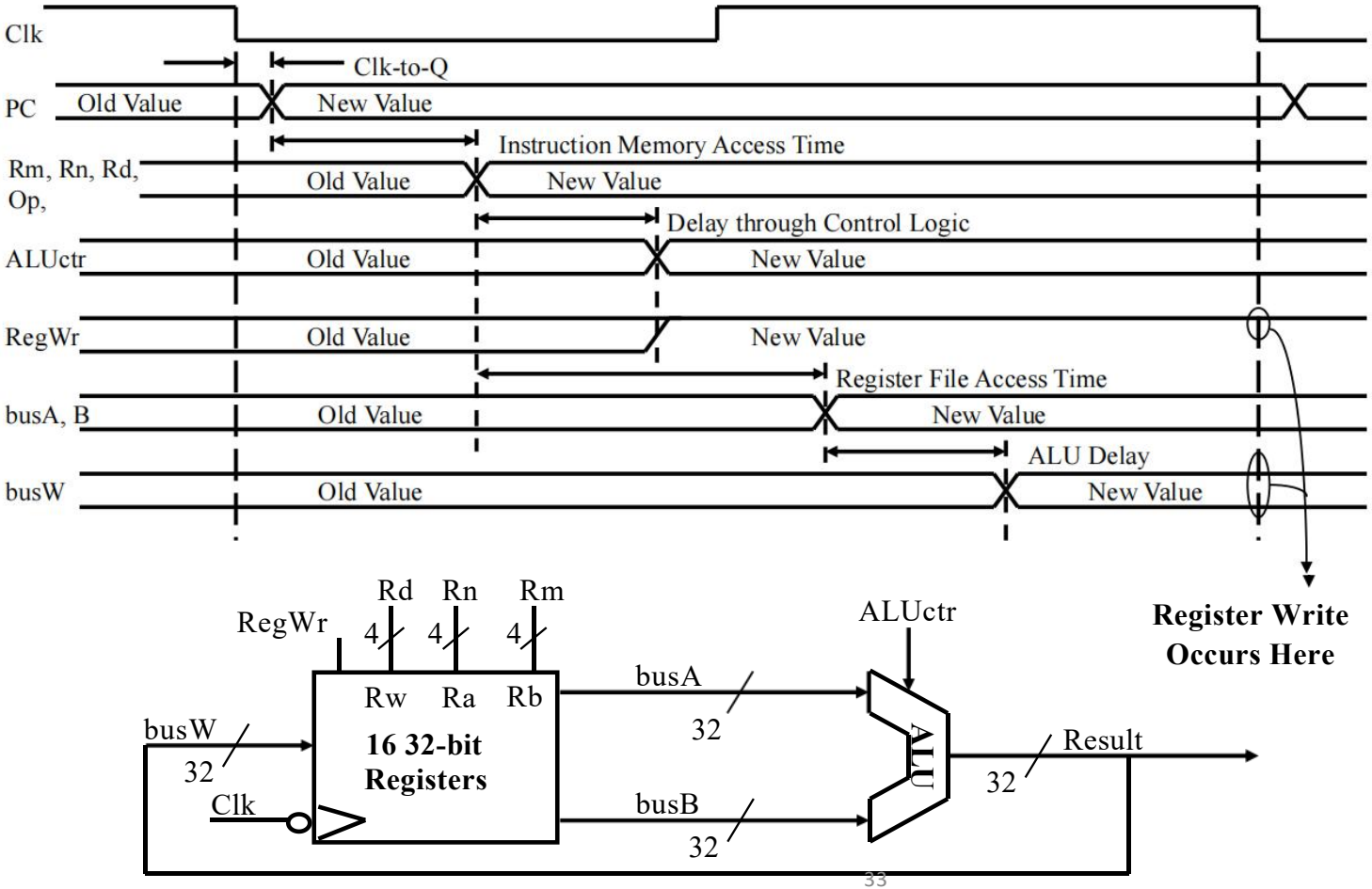


3b: 加法和减法

- **R[rd] <- R[rn] + R[rm]**　　Exemple: **add　rd, rn, rm**
  - Ra, Rb, 和 Rw 来自字段 rn, rm, 和 rd.
  - ALUctr 和 RegWr: 来自指令解码后的控制逻辑。

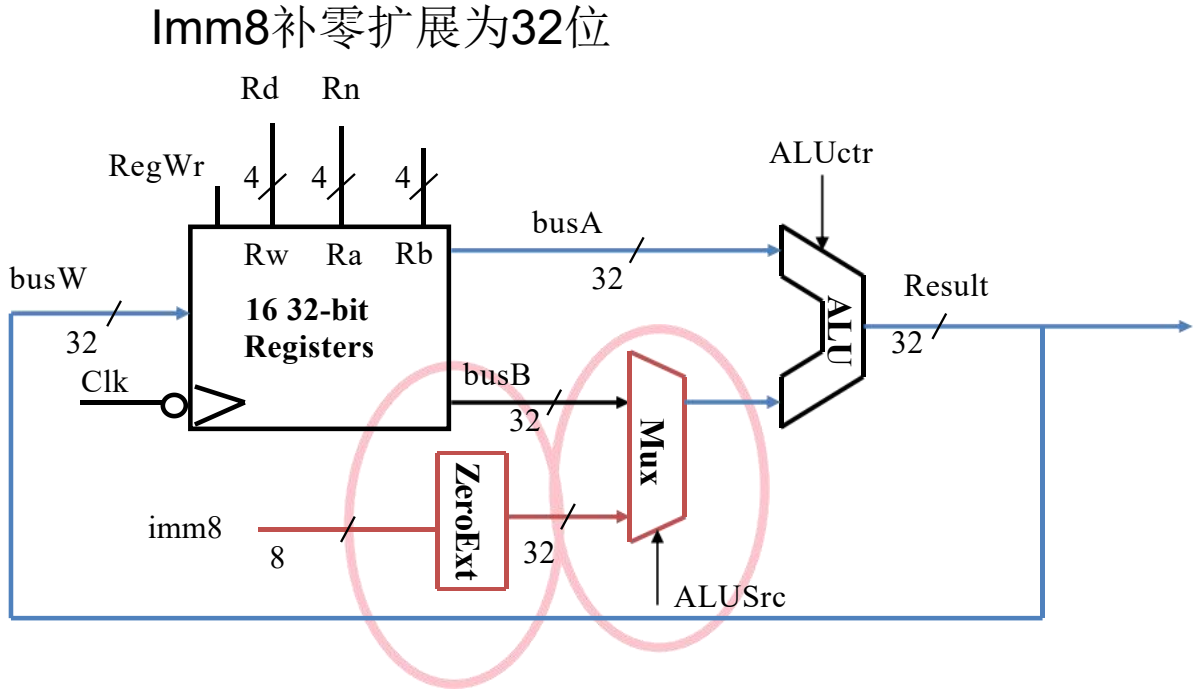


寄存器到寄存器时序


Clk
Clk-to-Q
PC
Old Value
New Value
Instruction Memory Access Time
Rm, Rn, Rd,
Op,
Old Value
New Value
Delay through Control Logic
ALUctr
Old Value
New Value
RegWr
Old Value
New Value
Register File Access Time
busA, B
Old Value
New Value
ALU Delay
busW
Old Value
New Value
Register Write
Occurs Here
Rd
Rn
Rm
RegWr
4
4
4
ALUctr
busA
Rw
Ra
Rb
busW
32
16 32-bit
Registers
32
ALU
Result
Clk
busB
32
32

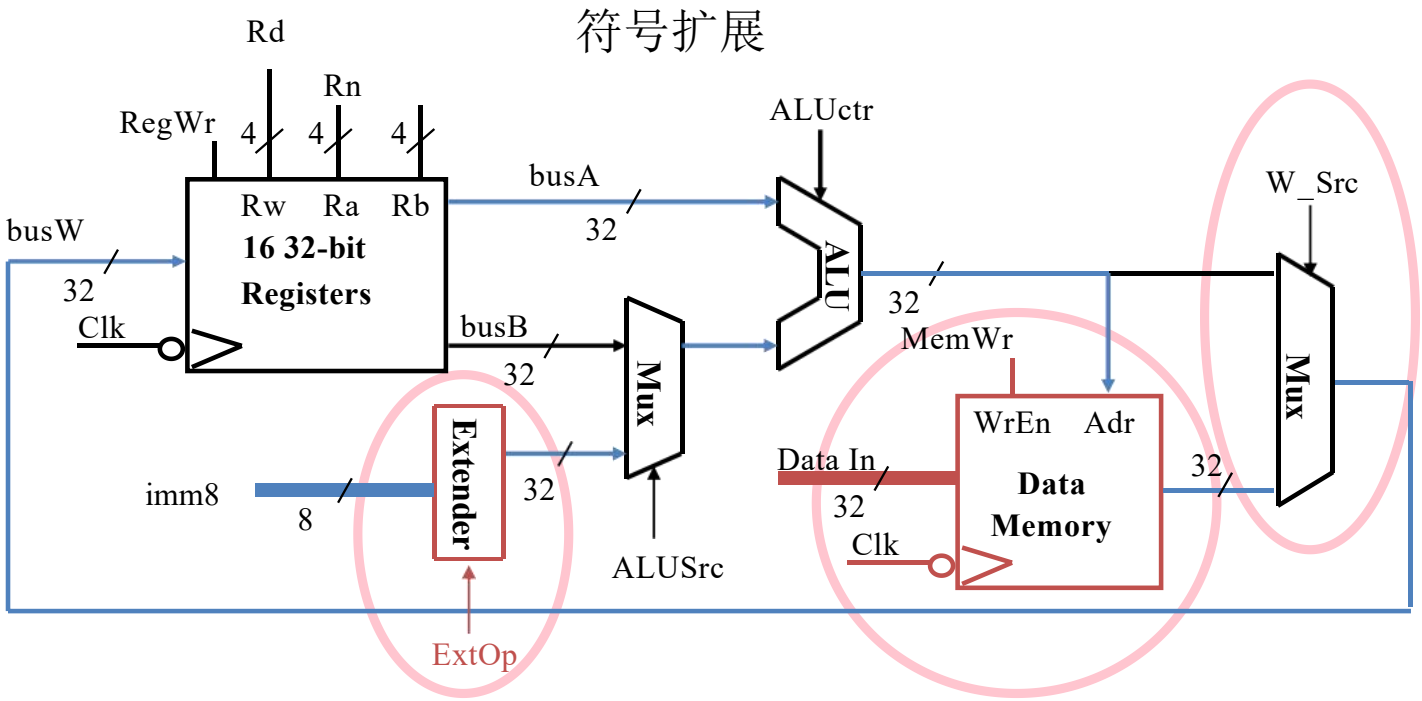
3c: 带立即数的逻辑运算

- R[rd] <- R[rn] + ZeroExt[imm8]



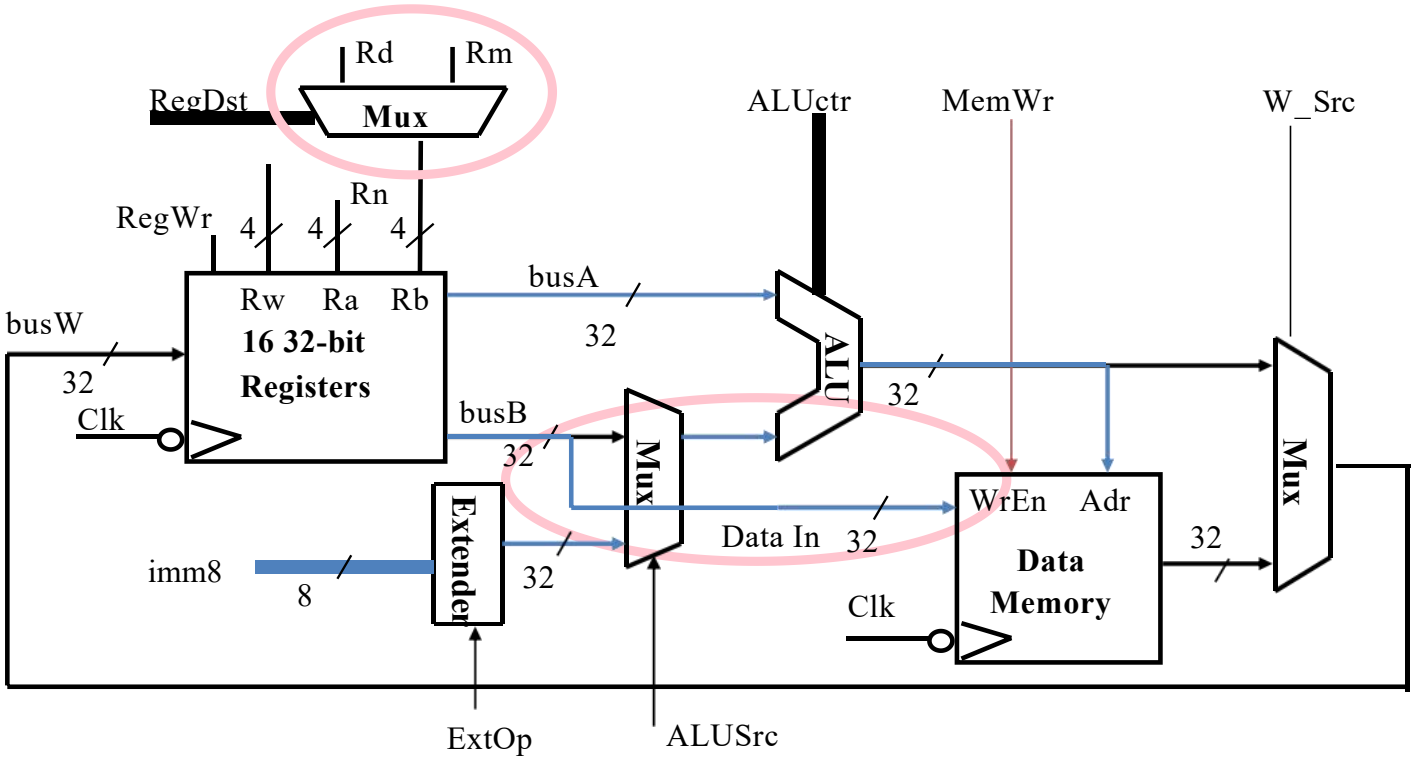
3d: 指令加载 Load

- R[rd] <- Mem[R[rn] + SignExt[imm8]]



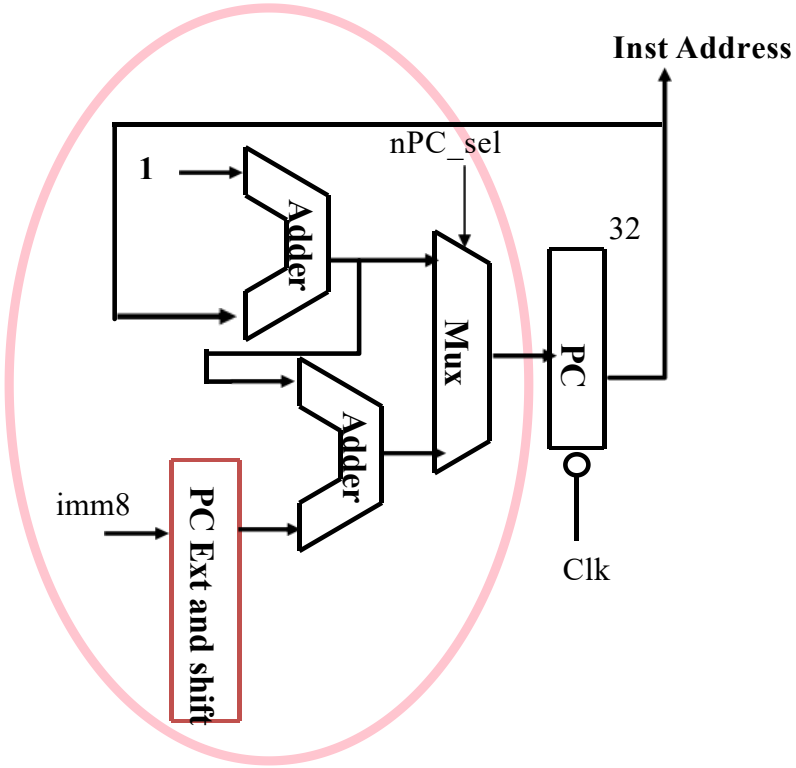
3e: 指令存储 Store

- Mem[R[rn] + SignExt[imm8]] <- R[rd]



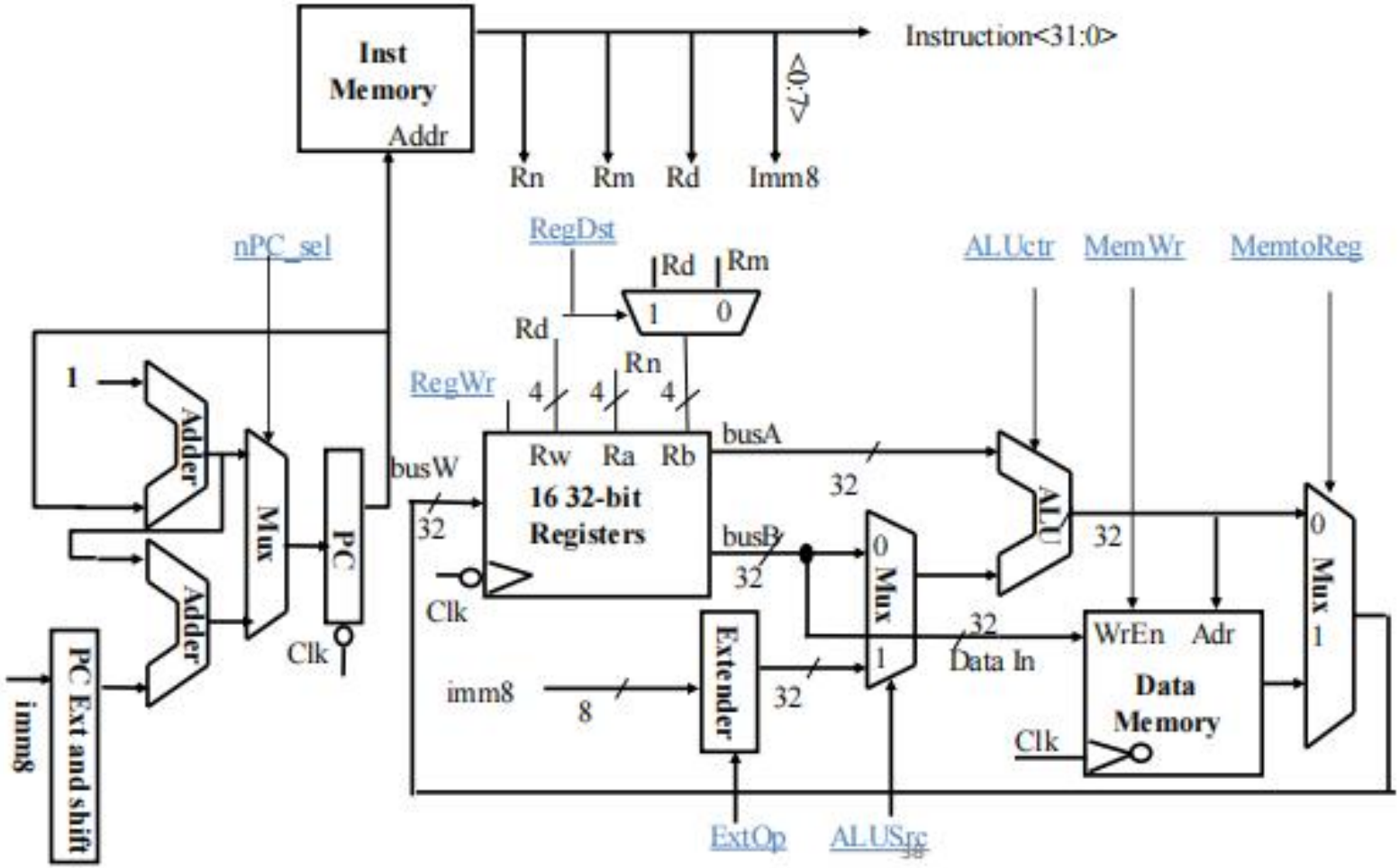
分支操作的数据路径

- CMP Rn, #Imm：CPSR 的标志 N = Rn – #Imm
- BLT 标签 ：Branch Lesser Than, 如果CPSR的标志 N = 1



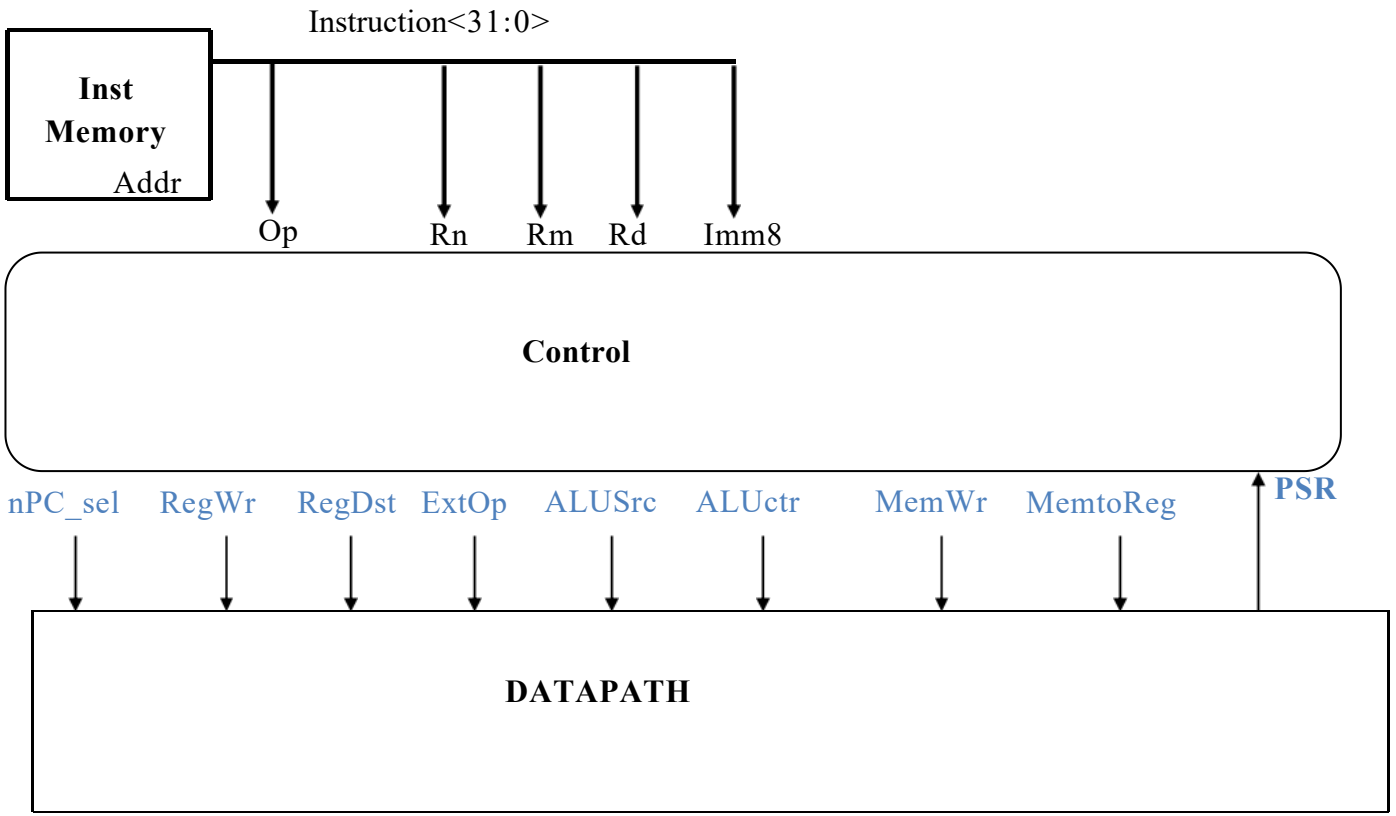
如果 nPC_sel = 0
PC = PC +1
如果 nPC_sel = 1
PC = PC + 1 + 立即数

组合：1个机器周期的数据路径



4/ 建立控制逻辑

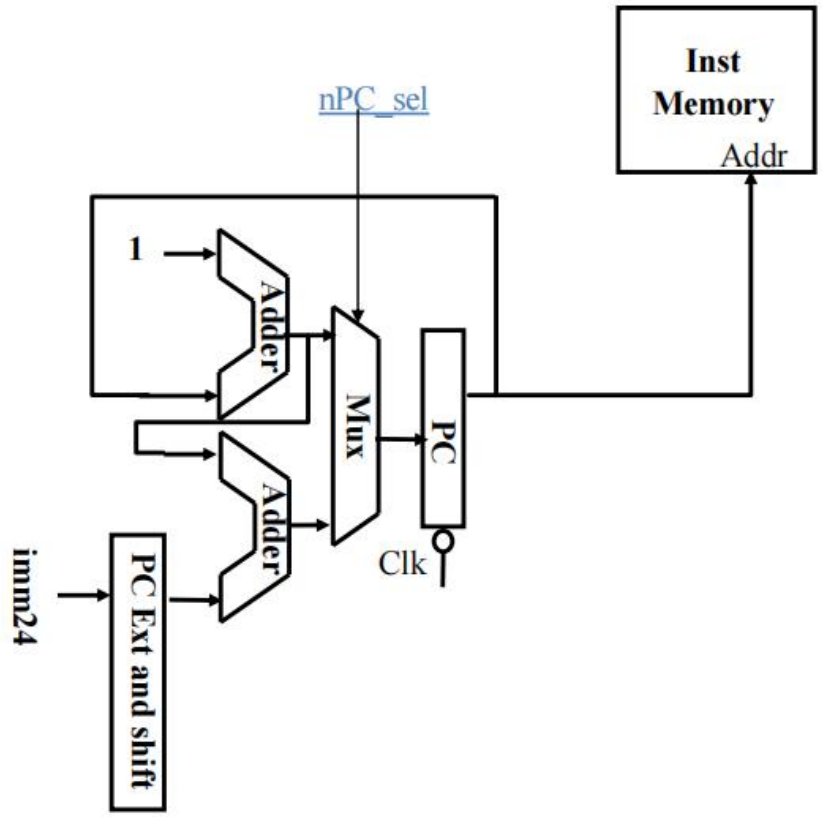
第4步：控制信号


Instruction<31:0>
Inst
Memory
Addr
Op
Rn
Rm
Rd
Imm8
Control
nPC_sel
RegWr
RegDst
ExtOp
ALUSrc
ALUctr
MemWr
MemtoReg
PSR
DATAPATH

控制信号的含义

- nPC_sel: 0 => PC <– PC + 1;

  1 => PC <– PC + 1 + (SignExt(imm24))



控制信号的含义

- ExtOp: “zero”, “sign”
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: “add”, “sub”, “mov”

- **MemWr:** 写入存储器
- **MemtoReg:** **1 => Mem, 0 => ALU out**
- **RegDst:** **0 => “rm”; 1 => “rd”**
- **RegWr:** 写入目标寄存器