哈爾濱工業大學

课程报告

课程名称：计算机组成原理

报告题目：RISC 处理器设计

所在院系：计算学部

所在专业：数据科学与大数据

学生姓名：董琦

学生学号：120L020701

选课时间：2022 年秋季学期

评阅成绩：

目录

一 问题综述

本实验的任务是设计一个简单的 RISC 处理器，该处理器是在给定的指令集下构建的，支持十条指令。假定主存可以在一个时钟周期内完成一次存取操作，而且可以和 CPU 同步工作。系统使用一个主存单元：指令读取和数据访问都使用同一组存储器。

处理器的指令字长为 16 位，包含 8 个 8 位通用寄存器 R0~R7，1 个 16 位的指令寄存器 IR 和 1 个 16 位的程序计数器 PC。取指令时，可以直接从主存中提取 16 位的指令信息，而进行数据访问时，与主存进行 8 位的数据交换。处理器的地址总线宽度是 16 位，数据总线宽度也是 16 位，无论是取指还是数据访问，使用同一组数据总线，只是数据信息的宽度不同。

处理器所支持的指令包括 LDA，STA，MOV，MVI，ADD，SUB，JZ，JMP，IN，OUT。其中仅有 LDA 和 STA 是访存指令，所有的存储器访问都通过这两条指令完成；ADD 和 SUB 是运算指令，MOV 和 MVI 是传数指令，他们都在处理器内部完成；JZ 是跳转指令，根据寄存器的内容进行绝对跳转；JMP 是无条件转移指令；IN 和 OUT 是输入输出指令，所有 I/O 端口与 CPU 之间的通信都由 IN 和 OUT 指令完成。

表 1　指令集合

指令	功能	备注
ADD Ri,Rj	Rj + Ri → Ri	加法
SUB Ri,Rj	Ri – Rj → Ri	减法
MOV Ri,Rj	Rj → Ri	寄存器传送
MVI Ri,X	X → Ri	立即数传送
STA Ri,X	Ri → [R7//X]	存数
LDA Ri,X	[R7//X] → Ri	取数
JZ Ri,X	If(Ri==0)then [R7//X] → PC	条件转移
JMP X	[R7//X] → PC	无条件转移
IN Ri,PORT	[PORT] → Ri	IO 输入
OUT Ri,PORT	Ri → [PORT]	IO 输出

二 指令格式

指令长度共 16 位。共 10 个操作，只需 4 位操作码；寄存器共 8 个，编码需三位，而若要区分不使用寄存器的情况，则需要 9 种情况，则需要四位编码；立即数与数据交换长度相同，为 8 位。

故一般格式为：前 4 位为操作码，后 12 位为地址码。

1. 若指令需要两个寄存器，则前 4 位为操作码（为与情况 2 区分，前俩位为 00），其后是两个四位的寄存器码，分别代表 Ri 和 Rj，最后四位多余，置为 0。如 ADD,SUB,MOV。

2. 若指令需要一个寄存器和一个立即数，则前四位为操作码，其后是一个四位的寄存器码，后 8 位为立即数。如 MVI,STA,LDA,JZ,IN,OUT。

3. 若指令不需要寄存器，只需要立即数，则前四位为操作码，中间四位全 1，表示不需要寄存器，后八位为立即数。如 JMP。

当不需要寄存器时，只有一种指令 JMP，故当 4~7 位为 1111 时即可知道需要读出后续的 8 位立即数。若有其他指令，则在 5~7 位另行设计即可。

表 2　指令编码

指令	0~3 位	4~7 位	8~11 位	12~15 位
ADD	0001	0XXX	0XXX	0000
SUB	0010	0XXX	0XXX	0000
MOV	0011	0XXX	0XXX	0000
MVI	0100	0XXX	XXXX	XXXX
STA	0101	0XXX	XXXX	XXXX
LDA	0110	0XXX	XXXX	XXXX
JZ	0111	0XXX	XXXX	XXXX
JMP	1000	1111	XXXX	XXXX
IN	1001	0XXX	XXXX	XXXX
OUT	1010	0XXX	XXXX	XXXX

三 微操作定义

1 前提假设

首先假定除通用寄存器 R0~R7、指令寄存器 IR、程序计数器 PC 外，我们的 CPU 中还有如下寄存器：

① valA,valB,valE：分别为 ALU 运算器的俩个输入值寄存器和一个输出值寄存器。
② valC：用于扩充地址的寄存器。
③ valP：用于更新 PC 计数器的寄存器。
④ MAR：16 位地址寄存器，用于告知内存将读写的地址。
⑤ MDR：8 位数据寄存器，用于与内存数据交换。
⑥ JZ 标志：若运算结果为 0，则 JZ 置 1。
⑦ R，W 信号：用于指示对内存的读、写使能。

然后我们将指令周期分为四个阶段：取指、译码、执行、写回。

① 取指：更新 PC，将 PC 中地址指示的指令取到 IR 中。
② 译码：将 IR 中指令解析，更新不同寄存器。
③ 执行：由于该指令集中指令要么访存，要么计算，或者俩者均无，所以可以将传统的访存与执行俩个阶段合并。
④ 写回：将中间寄存器的值写回到通用寄存器当中。

2 具体定义

此小节安排如下：2.1 为统一的取指周期的微指令，2.2~2.11 为十条指令的其余各周期微指令。此外，我们忽略 OP(IP)→CU 这一步骤，直接写译码过程。

2.1 取指周期

由于取指周期是所有指令共享的，故在这里统一定义如下：

valP → PC	**更新 PC 的值**
PC → MAR	**告诉内存要读取的地址**
1 → R	**内存读使能**
M(MAR) → MDR	**读取下一条指令到 MDR 中**
MDR → IR	**现行指令→IR**
PC+2 → valP	**写入 valP，+2 表示加俩个字节（16b）**

2.2 ADD

① 译码周期

[Ri] → valA **将 Ri 的值取出到 valA 中**

[Rj] → valB **将 Rj 的值取出到 valB 中**

② 执行周期

valA+valB→valE **由 ALU 计算结果**

③ 写回周期

valE→Ri **结果写回到 Ri 寄存器中**

2.3 SUB

① 译码周期

[Ri] → valA **将 Ri 的值取出到 valA 中**

[Rj] → valB **将 Rj 的值取出到 valB 中**

② 执行周期

valA-valB→valE **由 ALU 计算结果**

③ 写回周期

valE→Ri **结果写回到 Ri 寄存器中**

2.4 MOV

① 译码周期

[Ri] → valA **将 Ri 的值取出到 valA 中**

0 →valB **valB 取零值**

② 执行周期

valA+valB→valE **由 ALU 计算结果**

③ 写回周期

valE→Rj **结果写回到 Rj 寄存器中**

2.5 MVI

① 译码周期

X → valA **将立即数取出到 valA 中**

0 →valB **valB 取零值**

② 执行周期

valA+valB→valE **由 ALU 计算结果**

③ 写回周期

valE→Ri **结果写回到 Ri 寄存器中**

2.6 STA

① 译码周期

[Ri] → valA **将 Ri 的值取出到 valA 中**

X →valB **X 值取出到 valB 中**

[R7]→valC **取出地址高 8 位**

② 执行周期

[valC//valB]→MAR **将写入地址写到 MAR 中**

1→W **内存写使能**

valA→MDR **将写入内容写到 MDR 中**

MDR→M(MAR) **将 MDR 内容写到内存对应地址中**

③ 写回周期

无操作。

2.7 LDA

① 译码周期

X →valB **X 值取出到 valB 中**

[R7]→valC **取出地址高 8 位**

② 执行周期

[valC//valB]→MAR **将读取地址写到 MAR 中**

1→R **内存读使能**

M(MAR)→MDR **将读取内容写到 MDR 中**

③ 写回周期

MDR→Ri **将内容写回寄存器**

2.8 JZ

① 译码周期

[Ri] → valA **将 Ri 的值取出到 valA 中**

X →valB **X 值取出到 valB 中**

[R7]→valC **取出地址高 8 位**

② 执行周期

If(valA=0) JZ=1 **设置条件位**

③ 写回周期

If(JZ=0) [valC//valB]→valP **若 JZ 为 1 则设置跳转地址**

2.9 JMP

① 译码周期

X →valB **X 值取出到 valB 中**

[R7]→valC **取出地址高 8 位**

② 执行周期

无操作

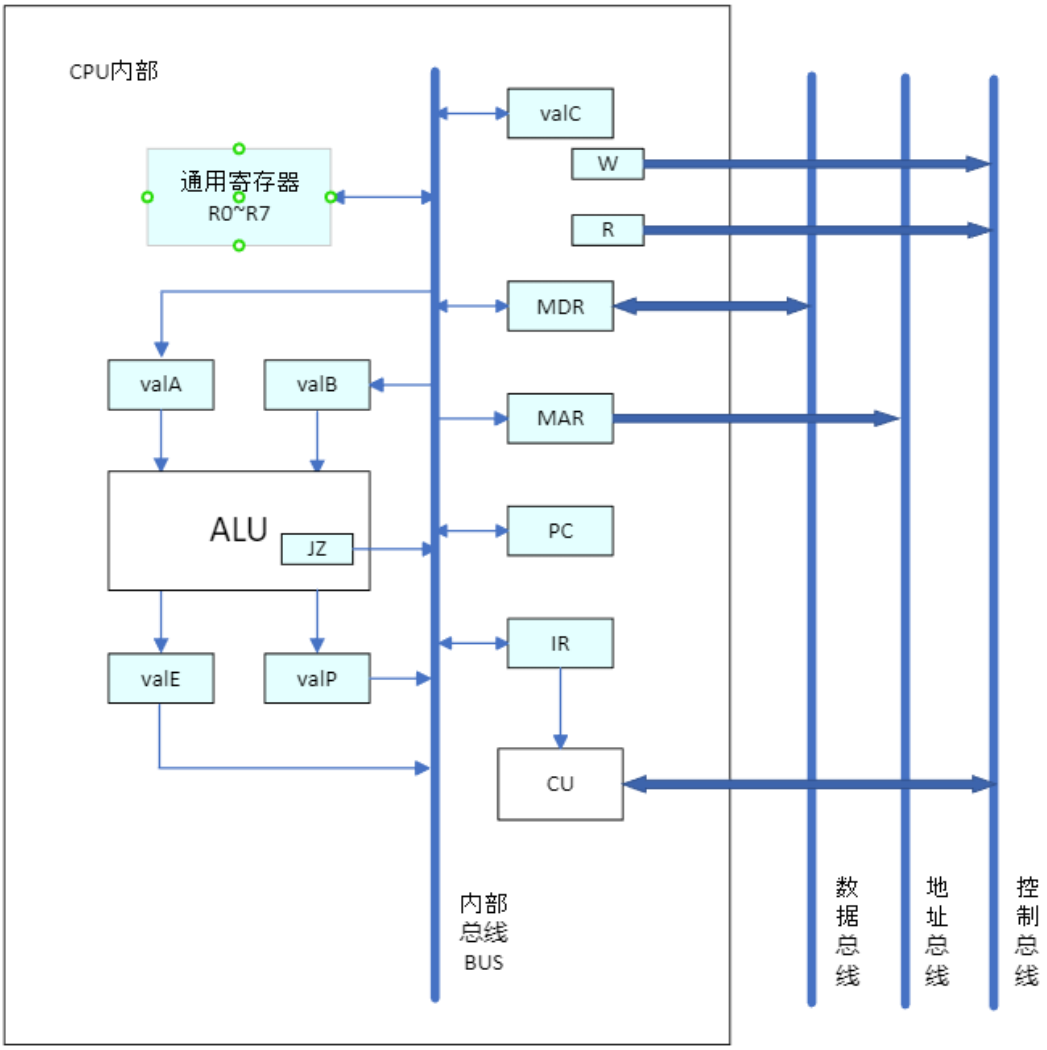
③ 写回周期

[valC//valB]→valP **设置跳转地址**

四 处理器结构设计

1 处理器结构框图

图 1 CPU 结构框图



2 处理器功能描述

① 通用寄存器 R0~R7：CPU 用于缓存数据

② valA,valB,valE：分别为 ALU 运算器的俩个输入值寄存器和一个输出值寄存器。

③ valC：用于扩充地址的寄存器。

④ valP：用于更新 PC 计数器的寄存器。

⑤ MAR：16 位地址寄存器，用于告知内存将读写的地址。

⑥ MDR：8 位数据寄存器，用于与内存数据交换。

⑦ JZ 标志：若运算结果为 0，则 JZ 自动置 1。

⑧ R，W 信号：用于指示对内存的读、写使能。

⑨ CU：解析 IR 中的指令，向各个部件发出控制信号。（图中省略连线）

⑩ ALU：运算器，做加减运算，并且根据结果设置 JZ 标志寄存器。

⑪ CPU 内部总线：用于传输 CPU 内部的数据。

五 微程序设计

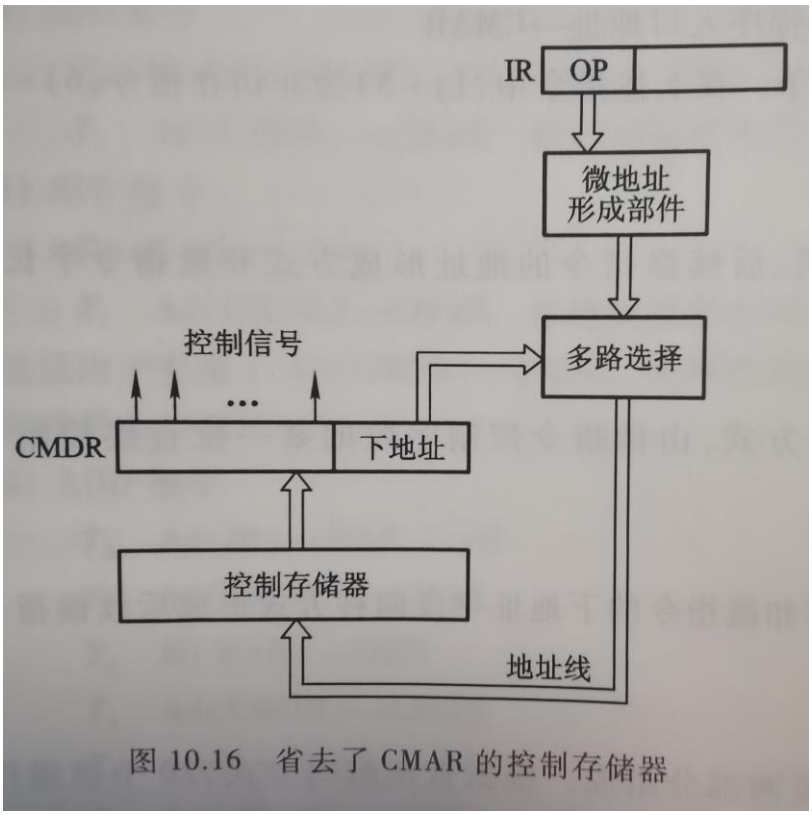
1 前提假设

我们使用微程序设计的方法来实现指令的执行。由于 valP 寄存器输出对象只有 PC，故可以在其之间设置一条单向通路，这样更新 PC 的数据流便不会占用总线资源，我们在后续微指令节拍划分时也忽略 valP→PC 这一步骤，由硬件自动完成。

对于寄存器文件，我们设置俩个读写双工端口。在读写寄存器时，首先由译码部件将读写的寄存器地址发送给寄存器文件，然后进行读写使能，最后执行微指令操作来读或写对应寄存器。俩个端口分别称为 REG0 和 REG1，分别对应指令中的 Ri 和 Rj，后续体现在微程序指令当中。

我们微程序的控制储存器采用图 2 的架构，使用一个多路选择控制器来替代 CMAR，从而省略了形成后续微指令地址的微指令和微操作。

图 2 控制存储器结构设计



2 节拍划分

2.1 取指周期

T_0 PC→MAR , 1→R

T_1 M(MAR)→MDR , PC+1→valP

T_2 MDR→IR , OP(IR)→微地址形成部件

2.2 ADD

① 译码周期

T_0 REG0→valA

T_1 REG1→valB

② 执行周期

T_0 valA+valB→valE

③ 写回周期

T_0 valE→REG0

2.3 SUB

① 译码周期

T_0 REG0→valA

T_1 REG1→valB

② 执行周期

T_0 valA-valB→valE

③ 写回周期

T_0 valE→REG0

2.4 MOV

① 译码周期

T_0 REG0→valA

T_1 0→valB

② 执行周期

T_0 valA+valB→valE

③ 写回周期

T_0 valE→REG1

2.5 MVI

① 译码周期

T_0 X→valA

T_1 0→valB

② 执行周期

T_0 valA+valB→valE

③ 写回周期

T_0 valE→REG0

2.6 STA

① 译码周期

T_0 REG0→valA

T_1 X→valB

T_2 REGA→valC （这里的 REGA 单指 R7，故使用一条单独指令）

② 执行周期

T_0 [valC//valB]→MAR 1→W

（将合成的地址传输到 MAR 中，使用一条单独指令）

T_1 valA→MDR

T_2 MDR→M(MAR)

③ 写回周期

无操作

2.7 LDA

① 译码周期

T_0 X→valB

T_1 REGA→valC

② 执行周期

T_0 [valC//valB]→MAR 1→R

T_1 M(MAR)→MDR

③ 写回周期

T_0 MDR→RGE0

2.8 JZ

① 译码周期

T_0 REG0→valA

T_1 X→valB

T_2 REGA→valC

② 执行周期

T_0 JZ=(val==0)? 1:0 (设置 JZ 位)

③ 写回周期

T_0 IF(JZ=0) [valC//valB]→valP

2.9 JMP

① 译码周期

T_0 X→valB

T_1 REGA→valC

② 执行周期

无操作

③ 写回周期

T_0 [valC//valB]→valP

3 微指令编码设计

上述共 38 条微指令，23 个微操作。我们使用**字段直接编码方式来编码微操作。**

首先可将微操作分为以下组：

① 使能

1→R , 1→W

② 取指

PC→MAR , PC+1→valP , MDR→IR

③ 译码

REG0→valA , X→valA

REG1→valB , X→valB , 0→valB

REGA→valC

④ 执行

valA+valB→valE , valA-valB→valE , JZ=(val==0)? 1:0

[valC//valB]→MAR , MDR→M(MAR) , M(MAR)→MDR

valA→MDR

⑤ 写回

valE→REG0 , valE→REG1 , MDR→REG1

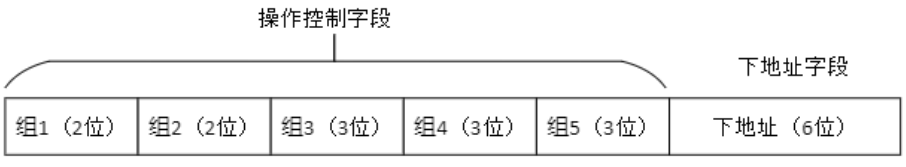
[valC//valB]→valP , IF(JZ=0) [valC//valB]→valP

组内指令单时钟周期互斥，所以我们可以对每一个组分别编码。

对于组①，共俩条指令，加上不发送，故需 2 位编码；对于组②，共 3+1=4 种情况，故需 2 位编码；对组③，共 6+1=7 种情况，故需 3 位编码；对组④，共 7+1=8 种情况，故需 3 位编码；对于组⑤，共 5+1=6 种情况，故需 3 位编码。所以我们指令操作控制字段共需 2+2+3+3+3=13 位编码。而共有 38 条微指令，所以需要 6 位下地址编码。

所以，我们的微指令字长为 19 位。微指令格式如下图：

图 3 微指令格式



对于各组内编码，详细格式如下：

① 00 无操作，01 1→R，10 1→W

② 00 无操作，01 PC→MAR，10 PC+1→valP，11 MDR→IR

③ 000 无操作，001 REG0→valA，010 X→valA，011 REG1→valB，100 X→valB，101 0→valB，110 REGA→valC

④ 000 无操作，001 valA+valB→valE，010 valA-valB→valE，011 JZ=(val==0)? 1:0，100 [valC//valB]→MAR，101 MDR→M(MAR)，110 M(MAR)→MDR，111 valA→MDR

⑤ 000 无操作，001 valE→REG0，010 valE→REG1，011 MDR→REG1，100 [valC//valB]→valP，101 IF(JZ=0) [valC//valB]→valP

表 2 微指令码点

微程序名称	微程序地址(八进制)	微指令（二进制代码）																	
		操作控制字段												顺序控制字段					
		①		②		③			④			⑤							
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
取指	00		1		1														1
	01			1					1	1								1	
	02			1	1										×	×	×	×	×	×
ADD	03							1										1		
	04						1	1										1		1
	05										1							1	1	
	06													1						
SUB	07							1									1			
	10						1	1									1			1
	11									1							1		1	
	12													1						
MOV	13							1									1	1		
	14					1		1									1	1		1
	15										1						1	1	1	
	16												1							

MVI	17						1								1				
	20					1		1							1				1
	21										1				1			1	
	22												1						
STA	23							1							1		1		
	24					1									1		1		1
	25					1	1								1		1	1	
	26	1							1						1		1	1	1
	27								1	1	1				1	1			
	30								1		1								
LDA	31					1									1	1		1	
	32					1	1								1	1		1	1
	33		1						1						1	1	1		
	34								1	1					1	1	1		1
	35												1	1					
JZ	36							1							1	1	1	1	1
	37					1								1					
	40					1	1							1					1
	41									1	1			1				1	
	42											1		1					
JMP	43					1								1			1		
	44					1	1							1			1		1
	45											1							