计算机组织与体系结构课程设计

基于基本模型机的堆栈设计与实现

一．实验目的

1．深入理解基本模型计算机的功能、组成知识；
2．深入学习计算机各类典型指令的执行流程；
3．学习微程序控制器的设计过程和相关技术，掌握 LPM_ROM 的配置方法。
4．在掌握部件单元电路实验的基础上，进一步将单元电路组成系统，构造一台基本模型计算机。
5．掌握微程序的设计方法，学会编写二进制微指令代码表。
6．设计关于数据栈的若干机器指令，实现栈的基本功能，并编写相应的微程序，上机调试，掌握计算机整机概念。
7．通过熟悉较完整的计算机的设计，全面了解并掌握微程序控制方式计算机的设计方法。

二．实验原理

1．本实验中，计算机数据通路的控制将由微过程控制器来完成，CPU 从内存中取出一条机器指令到指令执行结束的一个指令周期，全部由微指令组成的序列来完成，即一条机器指令对应一个微程序。
2．指令格式
2.1 微指令编码设计

表 1　24 位微指令编码定义

24	23	22	21	20	19	18	17	16	15 14 13	12 11 10	9 8 7	6	5	4	3	2	1
S3	S2	S1	S0	M	Cn	WE	A9	A8	A	B	C	uA5	uA4	uA3	uA2	uA1	uA0

- **微指令编码功能说明**

(1) uA5—uA0：微程序控制器的微地址输出信号，是下一条要执行的微指令的微地址。
(2) S3、S2、Sl、S0：由微程序控制器输出的 ALU 操作选择信号，以控制执行 16 种算术操作或 16 种逻辑操作中的某一种操作(见附表)。
(3) M：微程序控制输出的 ALU 操作方式选择信号端。M＝0 执行算术操作；M＝l 执行逻辑操作。
(4) Cn：微程序控制器输出的进位信号，Cn＝0 表示 ALU 运算时最低位无进位，Cn＝1 则表示有进位。
(5) WE：微程序控制器输出的 RAM 控制信号。当 WE＝0 时，为存储器读；当 WE＝1 时，为存储器写。
(6) A9、A8：译码后产生 CS0、CS1、CS2 信号，分别作为 SW_B、RAM、LED 的选通控制信号。

表 2　选通控制信号编码定义

A9	A8	选通控制信号
0	0	SW_B
0	1	RAM
1	0	LED
1	1	

(7) A 字段（15、14、13）——译码后产生与总线相连接的各单元的输入选通信号（见表 6-1）。

(8) B 字段（12、11、10）——译码后产生与总线相连接的各单元的输出选通信号（见表 6-1）。

(9) C 字段（9、8、7） ——译码后产生分支判断测试信号 P(1)~P(4)和 LDPC 信号（见表 6-1）。

表 3　A、B、C 各字段功能说明

A 字段				B 字段				C 字段			
15	14	13	选择	12	11	10	选择	9	8	7	选择
0	0	0		0	0	0		0	0	0	
0	0	1	LDRi	0	0	1	RS_B	0	0	1	P(1)
0	1	0	LDDR1	0	1	0		0	1	0	P(2)
0	1	1	LDDR2	0	1	1		0	1	1	P(3)
1	0	0	LDIR	1	0	0		1	0	0	P(4)
1	0	1	LOAD	1	0	1	ALU_B	1	0	1	LDAR
1	1	0	LDAR	1	1	0	PC_B	1	1	0	LDPC

2.1 控制台微程序设计

为了向 RAM 中装入程序和数据，检查写入是否正确，并能启动程序执行，还必须设计三个控制台操作微程序。

- 存储器读操作（KRD）：控制台 SWA、SWB 为“0 0”时，可对 RAM 连续手动读出操作。
- 存储器写操作（KWE）：控制台 SWA、SWB 为“0 1”时，可对 RAM 连续手动写操作。
- 启动程序（RP）：下载实验程序后，控制台 SWA、SWB 为“1 1”时，即可转入到微地址“01”号“取指令”微指令，启动程序运行。

表 4　控制台指令编码定义

SWB	SWA	控制台指令
0	0	读内存(KRD)
0	1	写内存(KWE)
1	0	
1	1	启动程序（RP)

3．机器指令编码设计

为了实现功能完善的堆栈操作，本次实验中设计了根据指定大小创建栈，连续压栈操作，连续出栈操作以及销毁栈操作。

在机器指令的设计过程中，一条机器指令对应一个微操作程序，因此首先应当确定四条机器指令分别对应的微程序设计。

3.1 堆栈模型结构设计

为了能够对冯诺依曼体系的模型机合理的使用主存空间，应当对指令和数据存储进行恰当的划分，保证在指令运行过程中不会出现数据写入冲突，因此在设计栈操作中，必须要控制栈在主存中的位置以及栈大小。

本次实验采用以下的模型结构设计：

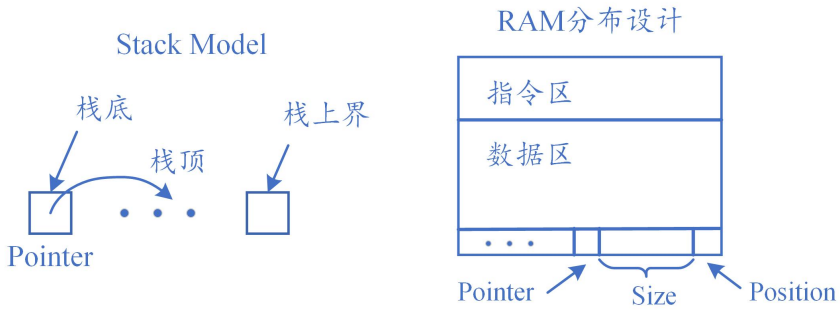

图 1 栈模型结构设计

3.2 堆栈机器指令设计

- CSTR——创建栈操作
  - 指令说明

CSTR　AR, SIZE

对应的机器指令说明在地址为 AR 处创建大小为 SIZE 的栈。

1. PC → AR
2. PC + 1
3. RAM → BUS
4. BUS →DR1
5. PC → AR
6. PC + 1
7. RAM → BUS
8. BUS → DR2
9. DR1 − DR2 → R0
10. R0 → AR
11. R0 → BUS
12. BUS → ARM

微程序中首先读取地址 AR 和栈大小 SIZE 到 DR1，DR2 中，通过减法操作计算出创建的栈底的地址，并在栈底存储单元中初始化栈顶为栈底地址，说明此时栈顶等于栈底，即栈大小为 0.

- PUSH——压栈操作
  - 指令说明

PUSH　DR

对应的机器指令说明在将数据 DR 处压入已经创建的栈中。

1. PC → AR
2. PC + 1
3. RAM → BUS
4. BUS →DR1
5. R0 → AR
6. RAM → BUS
7. BUS → DR2

栈未满：

8. R0 → AR
9. RAM → BUS
10. BUS → DR1
11. DR1 + 1 → BUS
12. BUS → ARM
13. RAM → BUS
14. BUS →AR
15. SW → BUS
16. BUS → RAM

栈满：

17. EXIT

微程序中首先读取 R0 寄存器中存储的栈底地址 AR，然后通过栈底地址获取栈顶的位置，紧接着进行栈满判断：通过将栈底和栈顶地址送入 DR1 与 DR2 中，在 ALU 中判断两者是否相等，如若不等说明此时栈未满，可以入栈，通过 P(3)分支跳转到入栈微程序代码段；如若相等说明此时栈已满，不能入栈，通过 P(3)分支跳转到退出微程序代码段。

其中对于入栈操作，首先获取栈顶位置，然后将其增 1 指向待写入存储单元地址，并将其重新写回栈底存储单元中以更新栈顶地址。接下来通过控制台微程序，读入待入栈操作数

到总线上，进一步再将总线上数据写入栈顶存储单元。

- POP——弹栈操作
  - 指令说明

POP　OUT

对应的机器指令说明在将栈顶数据 DR 弹出到 OUT 寄存器中并输出。

1. R0 → BUS
2. BUS → DR2
3. R0 → AR
4. RAM → BUS
5. BUS → DR1

栈非空：

8. DR1 → AR
9. RAM → BUS
10. BUS → DR2
11. DR2 → OUT
12. R0 → AR
13. DR1 − 1 → BUS
14. BUS → RAM

栈空：

15. EXIT

微程序中首先读取 R0 寄存器中存储的栈底地址 AR，然后将其送入到 DR2 中然后再通过栈底地址存储单元获取当前栈顶地址 AR，将其送入到 DR1 中。ALU 比较 DR1 与 DR2 是否相等，如若不等说明此时栈非空，可以进行弹栈操作，通过 P(3)分支跳转到弹栈微程序代码段；如若相等说明此时栈已空，不能进行弹栈操作，通过 P(3)分支跳转到退出微程序代码段。

其中对于弹栈操作，首先获取栈顶位置，然后获取其存储单元中的数据并将其送入到 DR2 中，通过 OUT 进行输出。紧接着，将 DR1 中栈顶位置减 1 重新写回到栈底位置对应的存储单元中。

- DSTR——销毁栈操作
  - 指令说明

DSTR　00

对应的机器指令说明将已创建的栈进行销毁，并以数据 00 进行填充表明主存中存储单元不代表任何数据。

1. PC → AR
2. PC + 1
3. RAM → BUS
4. BUS →DR1
5. R0 → AR
6. DR1 → BUS
7. BUS → RAM

微程序中首先读取主存中写入的立即数 00，然后将其送入到 DR1 中，再获取 R0 中存储的栈底地址，进一步将栈底地址对应的存储单元写入立即数 00 进行清零操作。

3.3 微指令编码设计

当机器指令对应的微程序设计完毕后，应将每条微指令代码化，表 4 即为设计的微指令按微指令格式转化而成的“二进制微代码表”。

表 5　二进制微代码表

微地址	微指令	S3	S2	S1	S0	M	CN	WE	A9	A8	A	B	C	UA5—UA0
0 0	018110	0	0	0	0	0	0	0	1	1	000	000	100	010000
0 1	01ED82	0	0	0	0	0	0	0	1	1	110	110	110	000010

0 2	00C048	0	0	0	0	0	0	0	0	1	100	000	001	001000
1 1	01EDAD	0	0	0	0	0	0	0	1	1	110	110	110	101101
1 2	01EDA3	0	0	0	0	0	0	0	1	1	110	110	110	100011
1 3	01B210	0	0	0	0	0	0	0	1	1	011	001	000	010000
1 4	0180CD	0	0	0	0	0	0	0	1	1	000	000	011	001101
1 5	018001	0	0	0	0	0	0	0	1	1	000	000	000	000001
1 6	01ED92	0	0	0	0	0	0	0	1	1	110	110	110	010010
2 0	01E211	0	0	0	0	0	0	0	1	1	110	001	000	010001
2 2	00A015	0	0	0	0	0	0	0	0	1	010	000	000	010101
2 5	01E216	0	0	0	0	0	0	0	1	1	110	001	000	010110
2 6	038A01	0	0	0	0	0	0	1	1	1	000	101	000	000001
3 5	01EA1E	0	0	0	0	0	0	0	1	1	110	101	000	011110
3 6	00B01F	0	0	0	0	0	0	0	0	1	011	000	000	011111
3 7	A90A20	1	0	1	0	1	0	0	1	0	000	101	000	100000
4 0	01E221	0	0	0	0	0	0	0	1	1	110	001	000	100001
4 1	F78A0B	1	1	1	1	0	1	1	1	1	000	101	000	001011
4 3	01A024	0	0	0	0	0	0	0	1	1	010	000	000	100100
4 4	01E225	0	0	0	0	0	0	0	1	1	110	001	000	100101
4 5	00B026	0	0	0	0	0	0	0	1	1	011	000	000	100110
4 6	0180E7	0	0	0	0	0	0	0	1	1	000	000	011	100111
4 7	018001	0	0	0	0	0	0	0	1	1	000	000	000	000001
5 5	00A02E	0	0	0	0	0	0	0	0	1	010	000	000	101110
5 6	01EDAF	0	0	0	0	0	0	0	1	1	110	110	110	101111
5 7	00B030	0	0	0	0	0	0	0	0	1	011	000	000	110000
6 0	619A31	0	1	1	0	0	0	0	1	1	001	101	000	110001
6 1	01E232	0	0	0	0	0	0	0	1	1	110	001	000	110010
6 2	038201	0	0	0	0	0	0	1	1	1	000	001	000	000001
6 7	01E238	0	0	0	0	0	0	0	1	1	110	001	000	111000

7 0	00A039	0	0	0	0	0	0	0	0	1	010	000	000	111001
7 1	078A3A	0	0	0	0	0	1	1	1	1	000	101	000	111010
7 2	01803B	0	0	0	0	0	0	0	1	1	000	000	000	111011
7 3	00E03C	0	0	0	0	0	0	0	0	1	110	000	000	111100
7 4	02000A	0	0	0	0	0	0	1	0	0	000	000	000	001010

3.4 微程序流程图设计

通过已设计的机器指令微程序以及编码的微指令，进一步表述机器指令流程如下图所示：

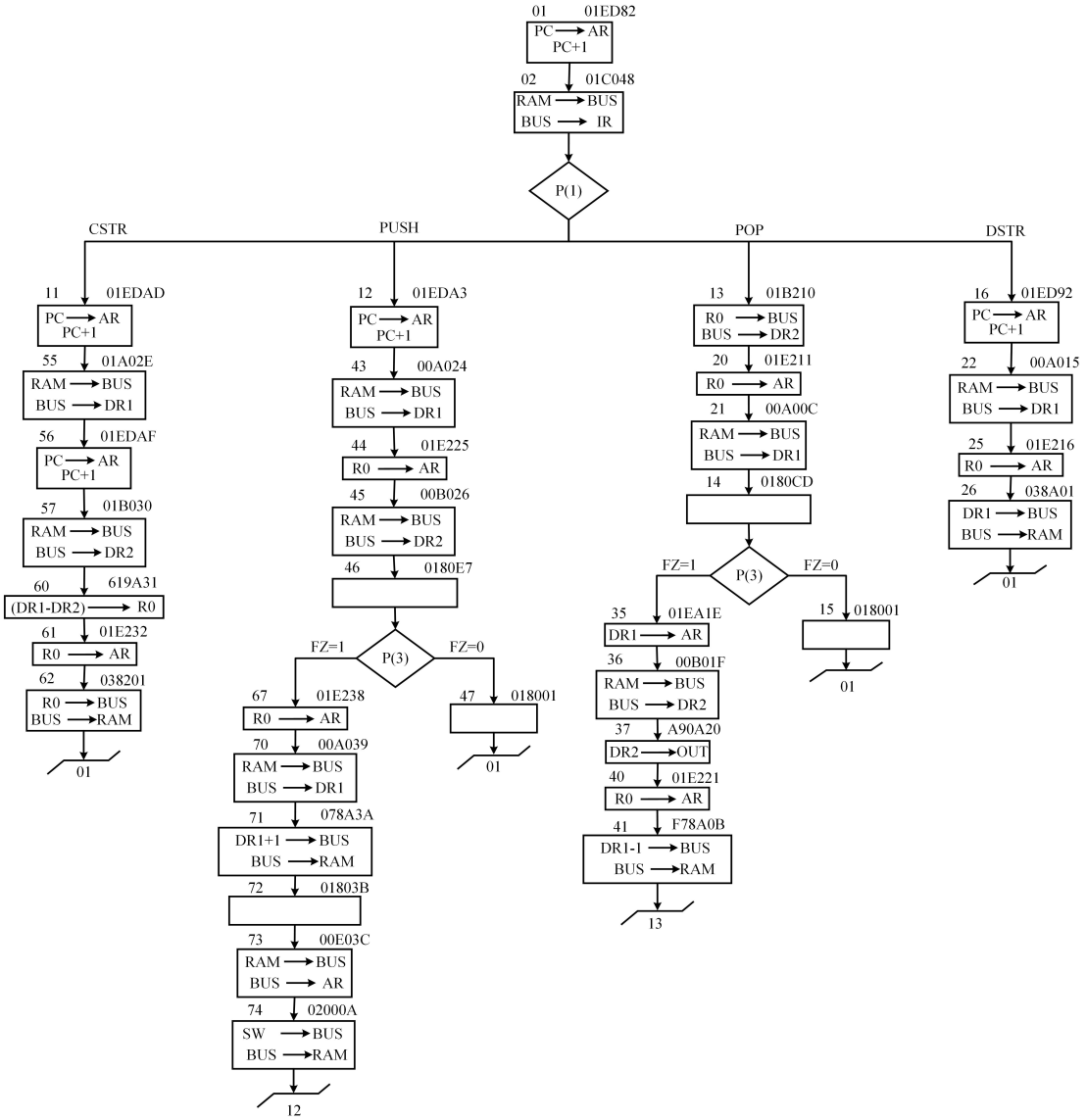

图 2 微程序流程图

4．RAM 存储器程序设计

利用已经设计好的机器指令，便可以编写一些程序执行栈的功能。本实验中将程序指令写入到 RAM 存储器中，程序执行时会顺序读取 RAM 中的指令，并且每条指令执行时均为单时钟周期。

本实验设计 RAM 中程序为，在存储地址为 FF 处创建一个大小为 3 的栈，连续压入 11，11，13 三个操作数，然后连续弹出压入的操作数，最后销毁已创建的数据栈。

表 6　实验程序设计

地址（二进制）	内容（二进制）	助记符	说明
00	10	CSTR	CSTR FF, 03
01	FF		
02	03		
03	20	PUSH	PUSH DATA
04	FF		
05	FF		
06	FF		
07	FF		
08	30	POP	POP DATA
09	60	DSTR	DSTR 00
0A	00		

三．实验步骤

1．实验结果说明

实验中 LCD 液晶显示屏可以用来显示模型机 CPU 中各组成单元的内容，其功能说明如下：

图 3　LCD 液晶显示屏

表 7　LCD 液晶显示屏功能说明

名称	作 用	名称	作 用

IN	输入单元 INPUT	DR1	暂存器 DR1
OUT	输出单元 OUTPUT	DR2	暂存器 DR2
ALU	算术逻辑单元	PC	程序计数器
BUS	内部数据总线	AR	地址寄存器
R0	寄存器 R0	RAM	程序/数据存储器
R1	寄存器 R1	IR	指令寄存器
R2	寄存器 R2	MC	微程序控制器

2．实验操作流程

本次实验在计算机上编译好代码文件，并随同模型 CPU 设计文件一同编译进 SOF 下载文件中，直接下载进入 FPGA。

(1) 按 1 次系统复位键 8，并置键 8 为高电平，使 CPU 允许正常工作；

(2) 控制开关（键 4、键 3）设置为 SWB、SWA=1,1，处于程序执行方式；

(3) 通过键 2、键 1 输入运算数据；

(4) 通过键 7 产生 1 个脉冲，执行 1 条微指令；

图 4 用在系统 EAB 读写工具对 FPGA 中的 ROM 和 RAM 进行观察和改写

五．实验结果

1．实验中 ROM 和 RAM 存储器配置

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	018110	01ED82	00C048	00E004	020001	009006	010201	00E00D
010	005181	01EDAD	01EDA3	01B210	0180CD	018001	01ED92	01EDAD
020	01E211	00A00C	00A015	018001	01E215	01E216	038A01	0180D8
030	01ED99	00E01A	00B01B	038A1C	01E21D	01EA1E	00B01F	A90A20
040	01E221	F78A0B	020023	00A024	01E225	00B026	0180E7	018001
050	01EA29	00B02A	A90A2B	01E22C	F78A01	00A02E	01EDAF	00B030
060	619A31	01E232	038201	038201	00B035	699A36	010201	01E238
070	00A039	078A3A	01803B	00E03C	02000A	000000	000000	000000
100	000000	000000	000000	000000	000000	000000	000000	000000
110	000000	000000	000000	000000	000000	000000	000000	000000
120	000000	000000	000000	000000	000000	000000	000000	000000
130	000000	000000	000000	000000	000000	000000	000000	000000
140	000000	000000	000000	000000	000000	000000	000000	000000
150	000000	000000	000000	000000	000000	000000	000000	000000
160	000000	000000	000000	000000	000000	000000	000000	000000
170	000000	000000	000000	000000	000000	000000	000000	000000
200	000000	000000	000000	000000	000000	000000	000000	000000
210	000000	000000	000000	000000	000000	000000	000000	000000
220	000000	000000	000000	000000	000000	000000	000000	000000

图 5 ROM 中微指令配置

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	10	FF	03	20	FF	FF	FF	FF
08	30	60	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00
28	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00
58	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00

图 6 RAM 中机器指令配置

2．实验结果分析

本次实验在地址为 FF 处创建的大小为 3 的栈如下图所示，FD 存储单元中的数据 FD 说明数据栈此时的栈顶在 FD 地址的存储单元，从内存中很容易读出 RAM[FD]=11，即此时压入的第一个数据 11。

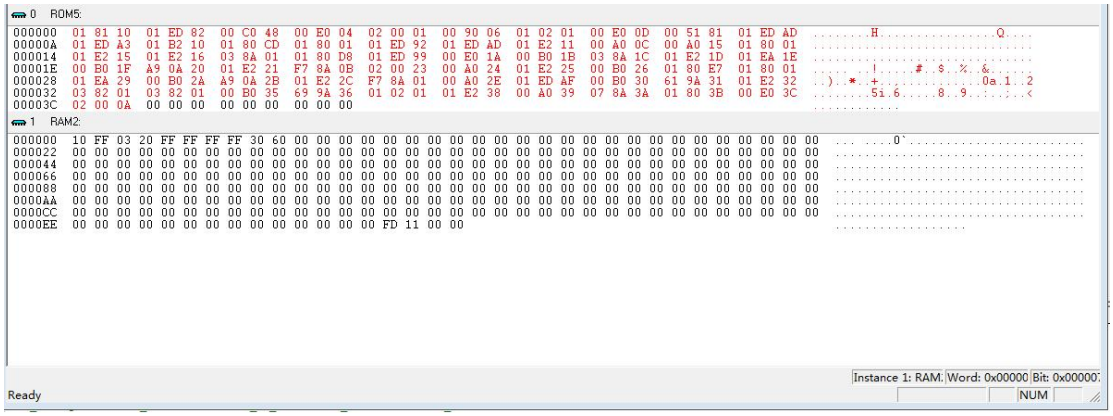

图 7 压栈后 RAM 数据分布

依次压入后两个数据，并进行连续 POP 操作后，可以看到主存中数据分布：

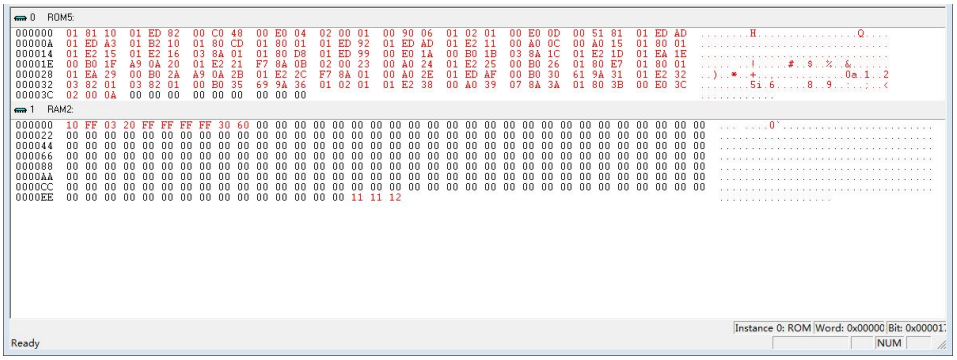

图 8 销毁栈后 RAM 数据分布

此使栈底位置的存储单元即 FC 处，其数据已经为被清零，表示此时栈已经被销毁。

五．实验中遇到的主要问题和解决思路

在本次实验中，为了实现数据栈较为完善的功能，需要对数据栈设定栈顶和栈底标志，以及最大上界，同时在操作上连续的压栈和弹栈要保证操作合法，即压栈时栈满检测和弹栈时非空检测。

对于这些问题，我们利用 R0 寄存器始终维护栈底地址，利用栈底存储单元维护栈顶地址，同时为了进行条件判断我们结合 P(3)分支进行了微指令的跳转设计，利用 ALU 的逻辑比较功能，实现了可定义的分支跳转。

同时为了能够便于创建数据栈，我们支持以指定大小创建数据栈，在实现中通过最大上界的单元地址和栈大小进行计算得出栈底的地址。