第十章 机器语言程序设计

本章重点

- 课程内容回顾
- 机器语言程序设计
- 示例

课程内容回顾

第六章 数据的机器级表示

第七章 数字逻辑电路

第八章 冯·诺依曼模型

第九章 指令集结构

指令集结构

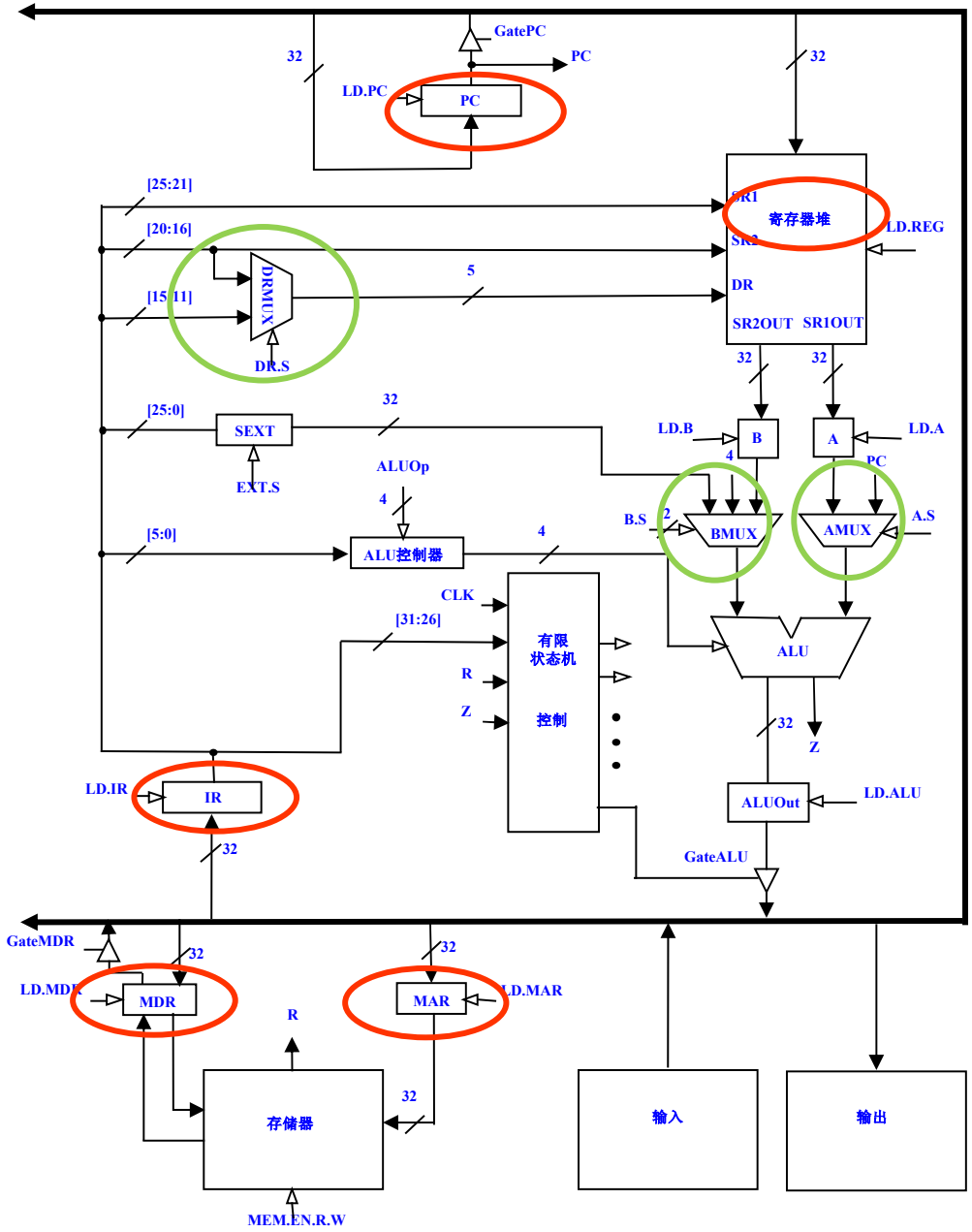
DLX
算术/逻辑运算指令
数据传送指令
控制指令
浮点指令

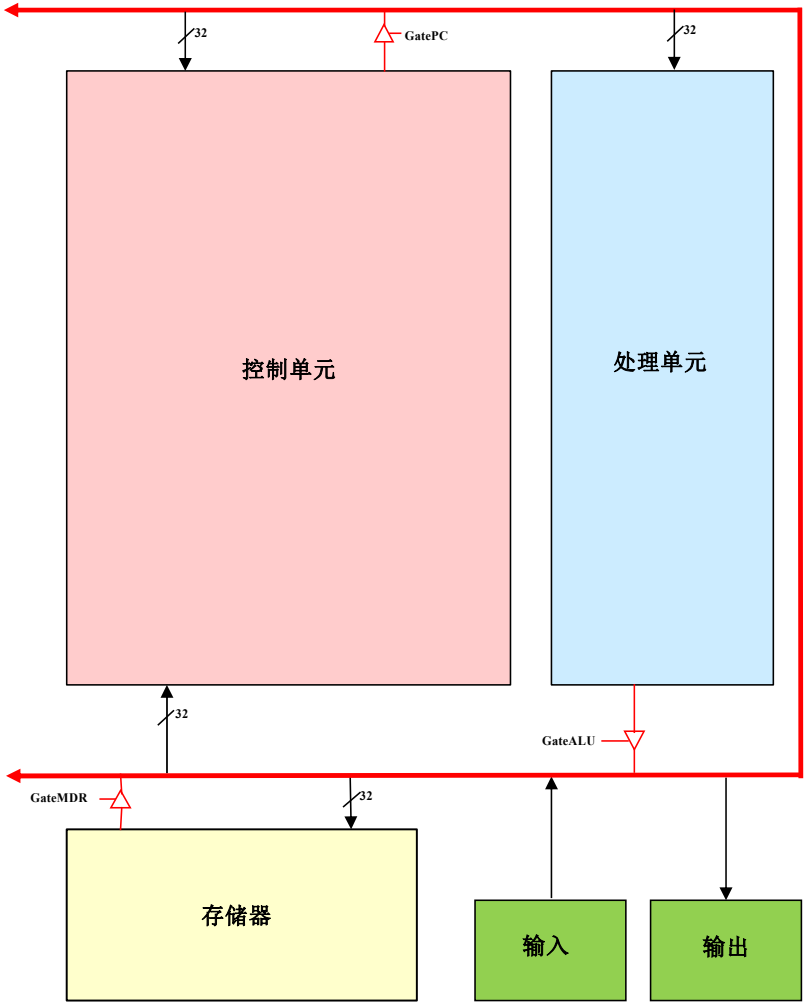
DLX
二进制补码整数
（8/16/32位）
单/双精度浮点数
（32/64位）

DLX
基址+偏移量

- 计算机能够执行的指令集合
  - 操作码：让计算机执行的操作
  - 操作数：每一步操作所需的数据
    - “数据类型”：操作数在计算机中的表示方式
    - “寻址模式”：如何计算操作数在存储器中的
  - 存储器
    - 地址空间：计算机存储单元的数量（2^n）
    - 寻址能力：每个存储单元存储信息的能力(m位)
  - 寄存器集（DLX包含32个整数寄存器）

不同的指令集结构规定的操作、操作数数据类型和寻址模式等是不同的。


GatePC
32
PC
LD.PC
PC
32
[25:21]
[20:16]
[15:11]
DRMUX
DR.S
5
SR1
SR2
寄存器堆
LD.REG
DR
SR2OUT
SR1OUT
32
32
[25:0]
SEXT
EXT.S
32
LD.B
B
A
LD.A
ALUOp
4
PC
B.S
2
BMUX
AMUX
A.S
[5:0]
ALU控制器
4
CLK
[31:26]
有限
状态机
R
Z
控制
ALU
32
Z
LD.IR
IR
32
ALUOut
LD.ALU
GateALU
GateMDR
32
32
LD.MDR
MDR
MAR
LD.MAR
R
存储器
32
MEM.EN.R.W
输入
输出

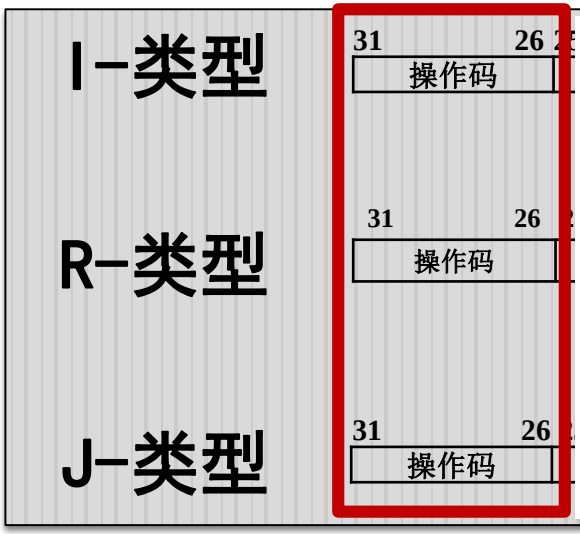

32
GatePC
32
控制单元
处理单元
32
GateALU
GateMDR
32
存储器
输入
输出

DLX 指令子集格式

	31 26	25 21	20 16	15 11	10 6	5 0
ADD	000000	SR1	SR2	DR	未用	000001
ADDI	000001	SR1	DR	Imm16		
SUB	000000	SR1	SR2	DR	未用	000011
SUBI	000011	SR1	DR	Imm16		
AND	000000	SR1	SR2	DR	未用	001001
ANDI	001001	SR1	DR	Imm16		
OR	000000	SR1	SR2	DR	未用	001010
ORI	001010	SR1	DR	Imm16		
XOR	000000	SR1	SR2	DR	未用	001011
XORI	001011	SR1	DR	Imm16		
LHI	001100	未用	DR	Imm16		
SLL	000000	SR1	SR2	DR	未用	001101
SLLI	001101	SR1	DR	Imm16		
SRL	000000	SR1	SR2	DR	未用	001110
SRLI	001110	SR1	DR	Imm16		
SRA	000000	SR1	SR2	DR	未用	001111
SRAI	001111	SR1	DR	Imm16		
SLT	000000	SR1	SR2	DR	未用	010000
SLTI	010000	SR1	DR	Imm16		
SLE	000000	SR1	SR2	DR	未用	010010
SLEI	010010	SR1	DR	Imm16		
SEQ	000000	SR1	SR2	DR	未用	010100
SEQI	010100	SR1	DR	Imm16		
LB	010110	SR1	DR	Imm16		
SB	010111	SR1	DR	Imm16		
LW	011100	SR1	DR	Imm16		
SW	011101	SR1	DR	Imm16		
BEQZ	101000	SR1	未用	Imm16		
BNEZ	101001	SR1	未用	Imm16		
J	101100	PCOffset26				
JR	101101	SR1	未用	未用		
JAL	101110	PCOffset26				
JALR	101111	SR1	未用	未用		
TRAP	110000	Vector26				

DLX指令操作类型

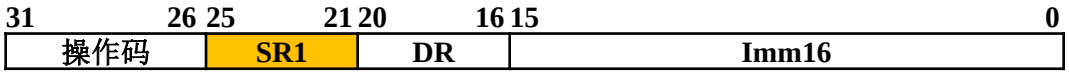
- 由指令的[31:26]位定义，64种指令类型
- R类型
  - 指令的[31:26]位为000000
  - [5:0]位定义了函数，有64种可能的函数
- 只定义了91条指令



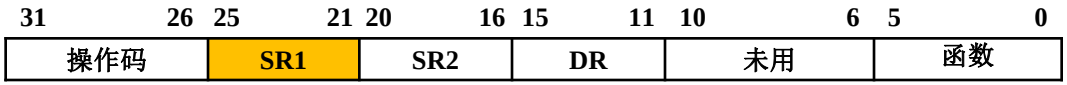
- 按照功能分为四种类型
  - 算术/逻辑运算指令
    - 处理整数信息
  - 数据传送指令
    - 在存储器和寄存器之间传送数据
    - 在寄存器/存储器和输入/输出设备之间传送数据
  - 控制指令
    - 改变指令被执行的顺序
  - 浮点指令
    - 处理浮点数信息

DLX指令：第一个源操作数

- 来自寄存器
  - 用5位编码标识
  - [25:21]，SR1
- I-类型



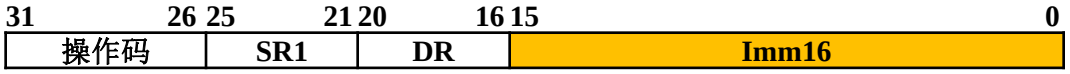
- R-类型



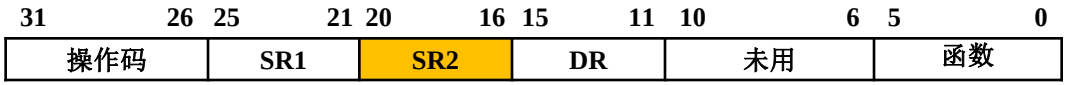
为什么用5位？

DLX指令：第二个源操作数

- I-类型，Immediate
  - [15:0]，直接获得
  - 立即数

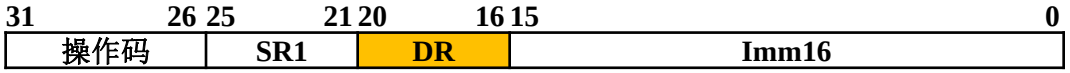


- R-类型，Register
  - [25:21]，SR2

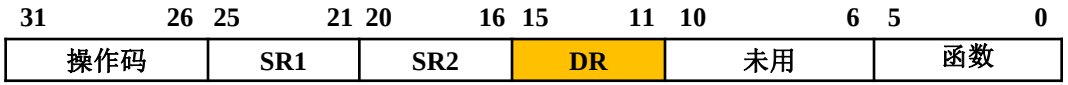


DLX指令：目标操作数

- I-类型，Immediate
  - [20:16]，DR

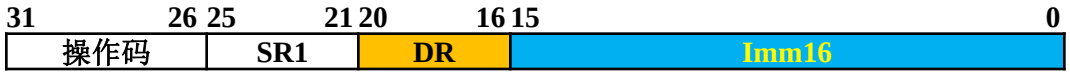


- R-类型，Register
  - [15:11]，DR

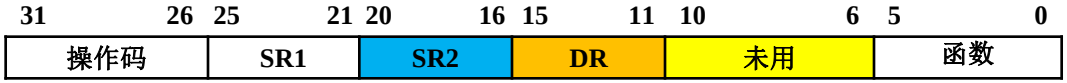


I-类型运算指令

- 第二个源操作数
  - 来自于指令[15:0]进行符号扩展得到的32位整数，即立即数
- 目标操作数
  - 来自于指令[20:16]所标识的寄存器中



R-类型运算指令

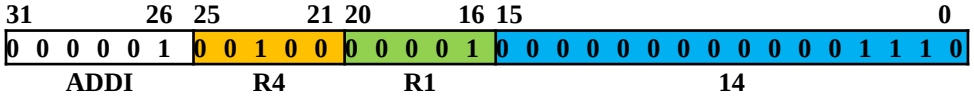


- [31:26]位为000000, [5:0]位定义了函数
- 第一个源操作数
  - 来自于指令[25:21]所标识的寄存器中
- 第二个源操作数
  - 来自于指令[20:16]所标识的寄存器中
- 目标操作数
  - 来自于指令[15:11]所标识的寄存器中

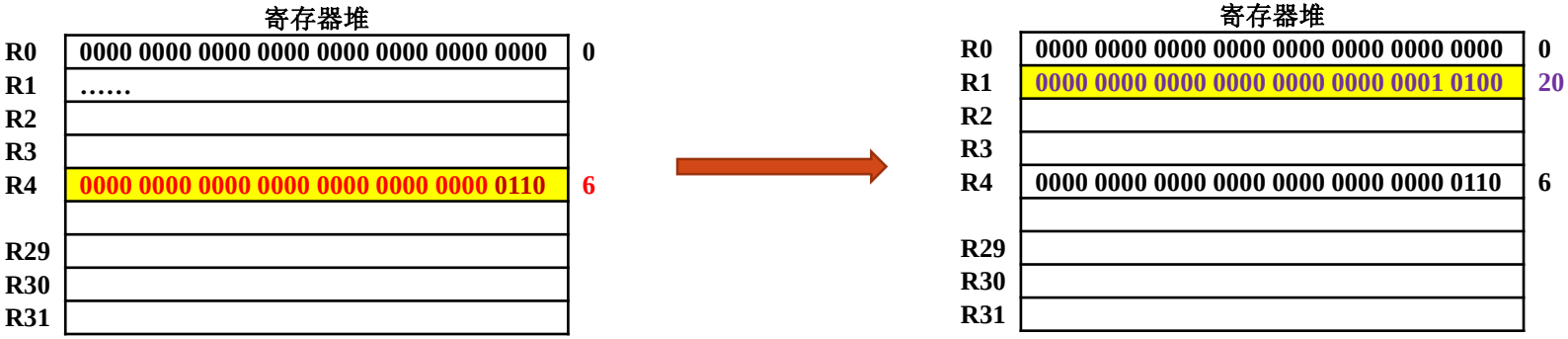
算术/逻辑运算指令

- 对整数进行处理
- 37个算术逻辑运算指令：加、减、乘、除、与、或、异或、移位、比较、加载高位立即数等
- 除加载高位立即数指令（LHI）外，其他运算指令执行的都是二元运算
  - 两个源操作数（即待运算的数据）
    - 来自通用寄存器或从指令中直接获得
  - 一个目标操作数（运算执行后的结果）
    - 存储于通用寄存器中

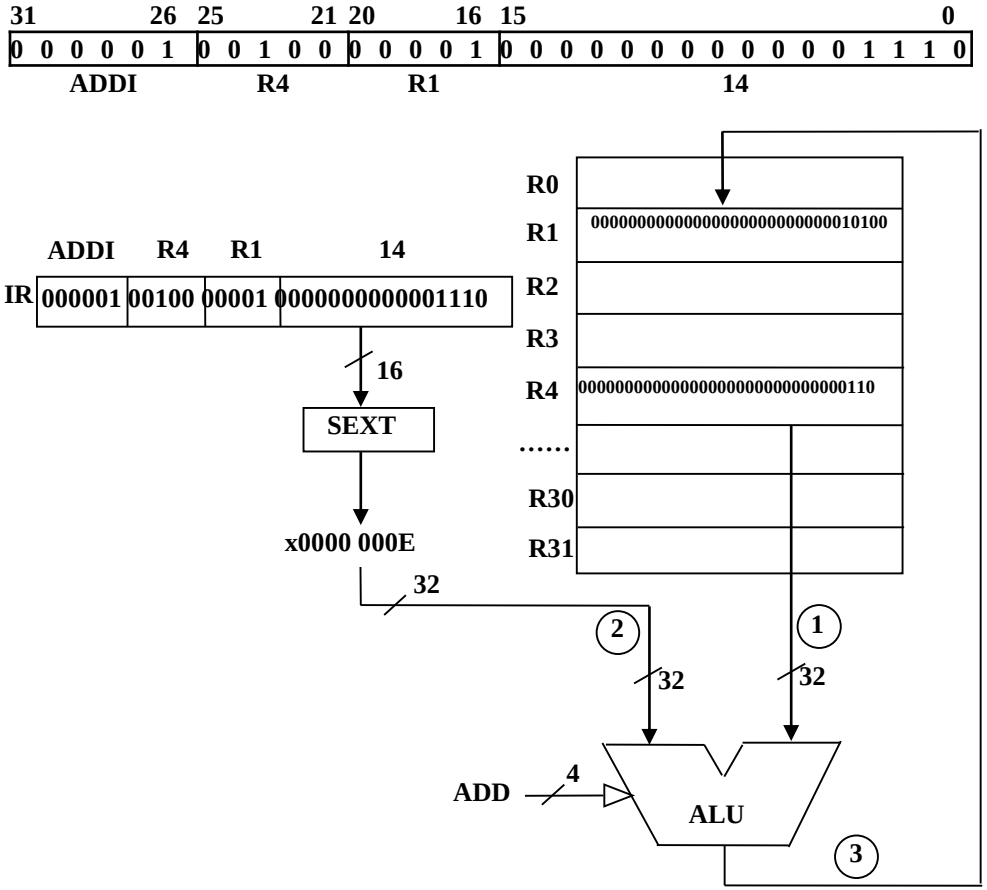
ADDI（算术运算指令）



- ADD代表加，I代表立即数（Immediate）
  - 第一个操作数R4
  - 第二个源操作数在指令中
    - [15:0]位符号扩展（SEXT）
  - 目标操作数写入R1

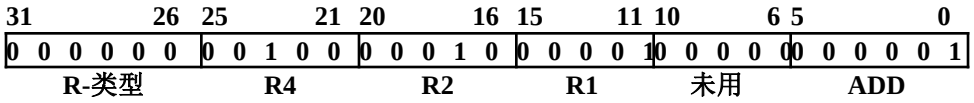


ADDI （算术运算指令）



哪些整数可以用作立即数？

ADD



- 操作码000000，R-类型
  - [5:0]为000001，ADD函数

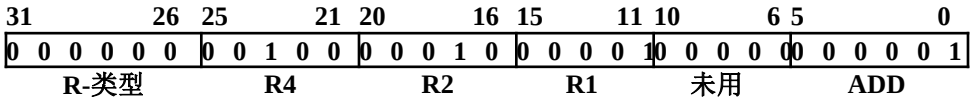
寄存器堆

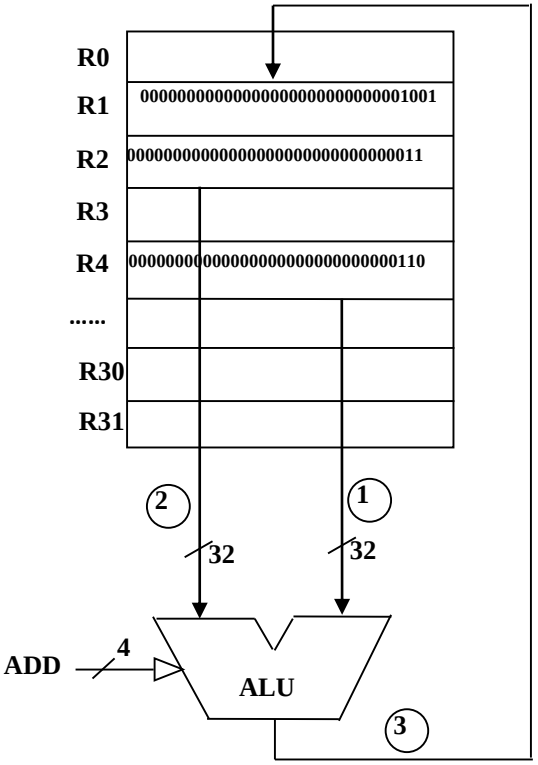
R0	0000 0000 0000 0000 0000 0000 0000 0000	0
R1	******	*
R2	0000 0000 0000 0000 0000 0000 0000 0011	3
R3		
R4	0000 0000 0000 0000 0000 0000 0000 0110	6
R29		
R30		
R31		

寄存器堆

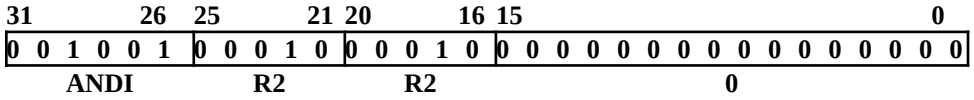
R0	0000 0000 0000 0000 0000 0000 0000 0000	0
R1	0000 0000 0000 0000 0000 0000 0000 1001	9
R2	0000 0000 0000 0000 0000 0000 0000 0011	3
R3		
R4	0000 0000 0000 0000 0000 0000 0000 0110	6
R29		
R30		
R31		

ADD

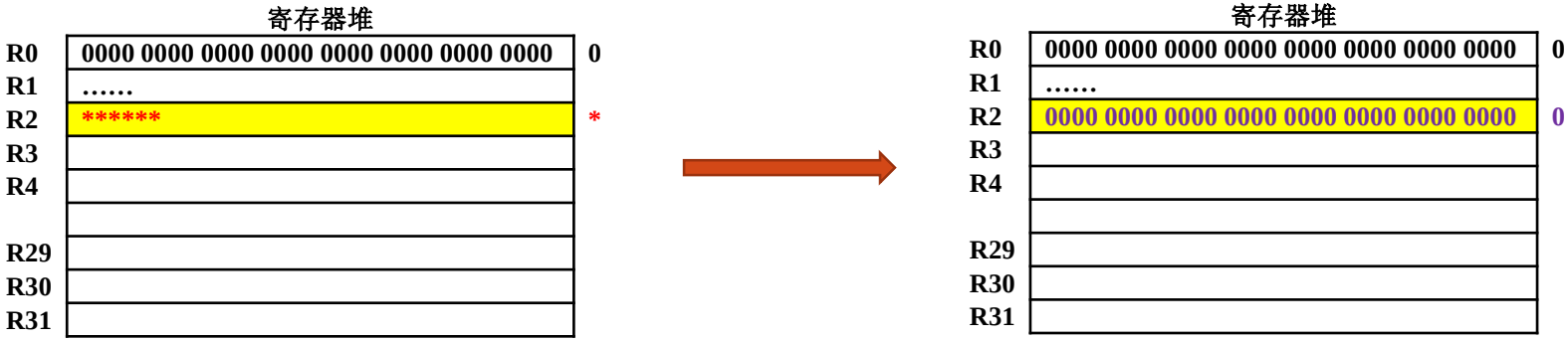

31 26 25 21 20 16 15 11 10 6 5 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1
R-类型
R4
R2
R1
未用
ADD


R0
R1
00000000000000000000000000001001
R2
00000000000000000000000000000011
R3
R4
00000000000000000000000000000110
.....
R30
R31
2
1
32
32
ADD
4
ALU
3

ANDI（逻辑运算指令）



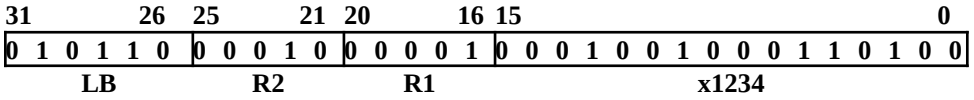
- 指令执行结果：寄存器R2被清空
  - R2←(R2) AND 0
  - 结果，R2的32位全部为0



数据传送指令

- 存储器和通用寄存器之间
- 加载（load）：将数据从存储器移动到寄存器的过程
- 存储（store）：将数据从寄存器移动到存储器的过程
- LB和SB：加载和存储一个8位的字节，在一个存储单元和一个寄存器之间传送数据
- LW和SW：加载和存储一个32位的字，在4个连续的存储单元和一个寄存器之间传送数据

LB （数据传送指令）



- 基址+偏移量
  - （R2）+ x0000 1234
  - x56781234
- 符号扩展到32位

地址	
x5678 1234	0000 1111
x5678 1235	******
x5678 1236	******
x5678 1237	******

寄存器堆

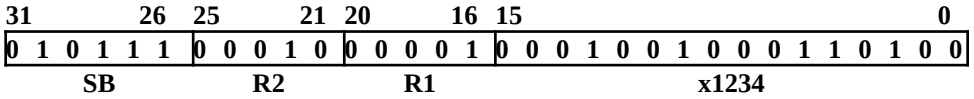
R0	0000 0000 0000 0000 0000 0000 0000 0000	0
R1	******	*
R2	0101 0110 0111 1000 0000 0000 0000 0000	x5678 0000
R3		
R4		
R29		
R30		
R31		

寄存器堆

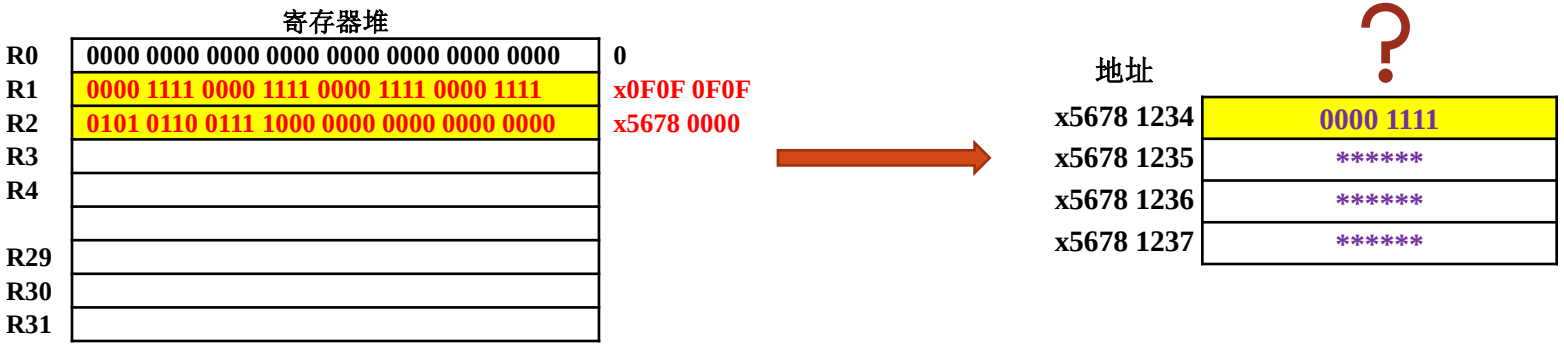
R0	0000 0000 0000 0000 0000 0000 0000 0000	0
R1	0000 0000 0000 0000 0000 0000 0000 1111	15
R2	0101 0110 0111 1000 0000 0000 0000 0000	
R3		
R4		
R29		
R30		
R31		

?

SB （数据传送指令）



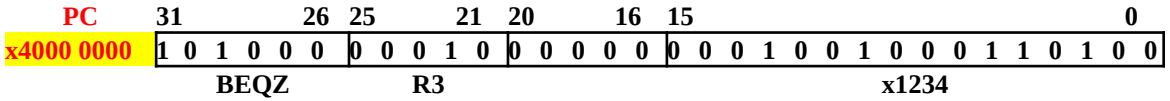
- 基址+偏移量
  - （R2）+ x0000 1234
  - x56781234
- R1中数值低8位（最低有效字节）



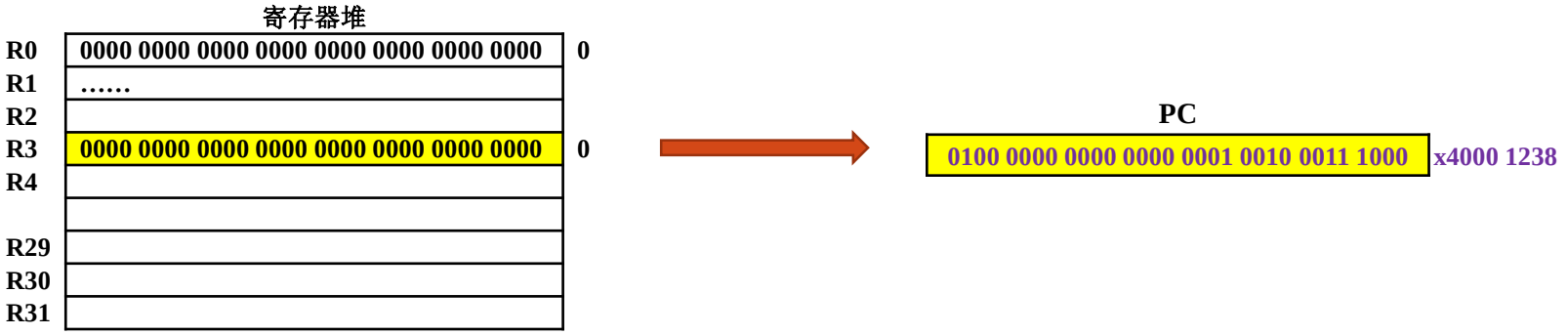
控制指令

- 改变被执行的指令的顺序
- DLX有10条指令能使顺序流被打破
  - 条件分支：BEQZ、BNEZ指令
  - 无条件跳转：JR、J指令
  - TRAP指令
  - 子例程（有时称为函数）调用
  - 从异常/中断返回

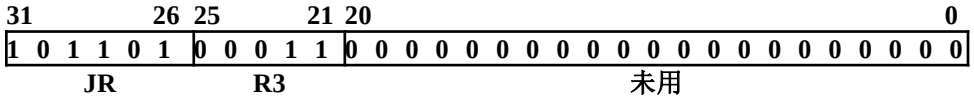
BEQZ指令（控制指令）



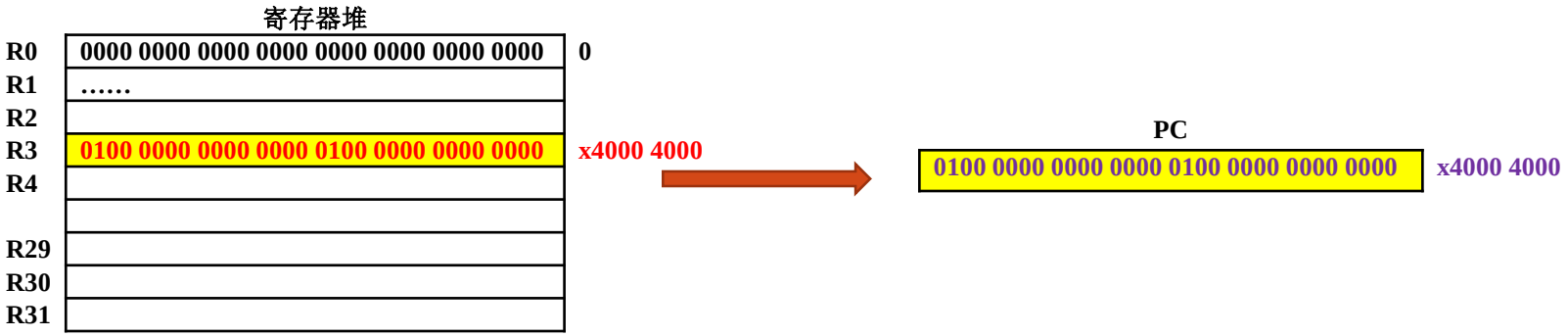
- PC ← PC + 4 + SEXT(Imm16)
  - 实际PC ← PC + x0000 0004 + x0000 1234



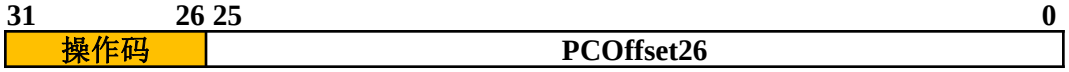
JR指令（控制指令）



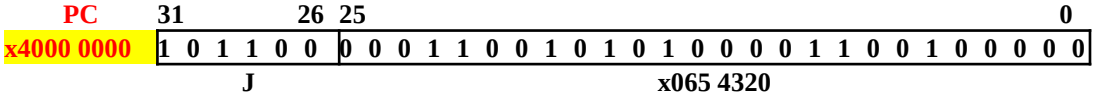
- 寄存器跳转（Jump Register）
- I-类型
- [20：0]位未用，设为0
- [25:21]位的寄存器
  - 包含下一条将要被执行的指令地址
- PC ← (R3)



J指令（控制指令）



- 跳转（Jump）
- J-类型
- PC ← PC+4+SEXT(PCOffset26)
- 订正：边界对齐，x4320



思考

- 为什么需要这两种不同类型的指令？

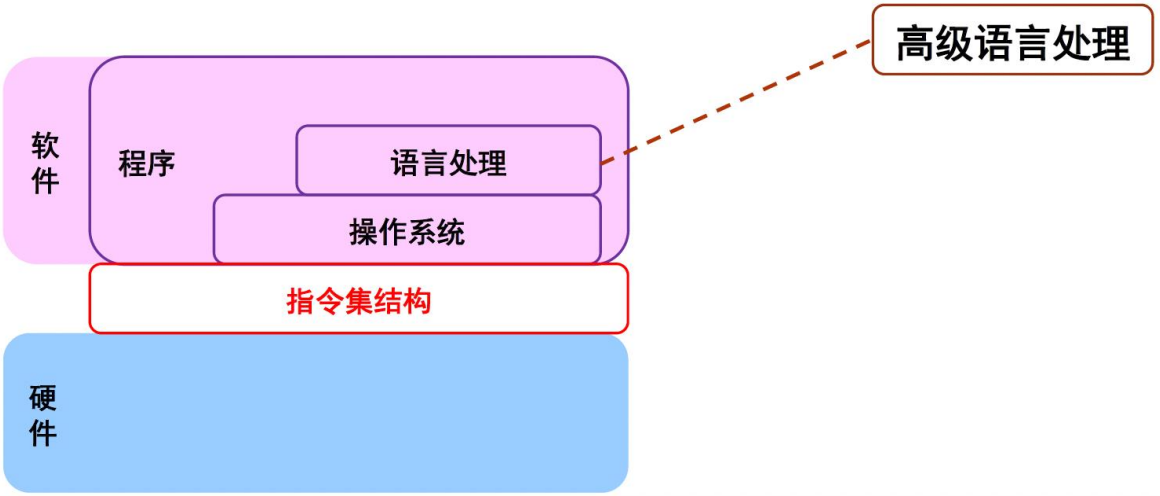
	I-类型	R-类型
灵活性	内存访问、带有常数的操作	寄存器之间的操作
性能	相对较慢，需要访问内存	相对较快，寄存器级别
内存访问	允许访问，数据加载和存储	不允许

本章重点

- 课程内容回顾
- 机器语言程序设计
- 实例

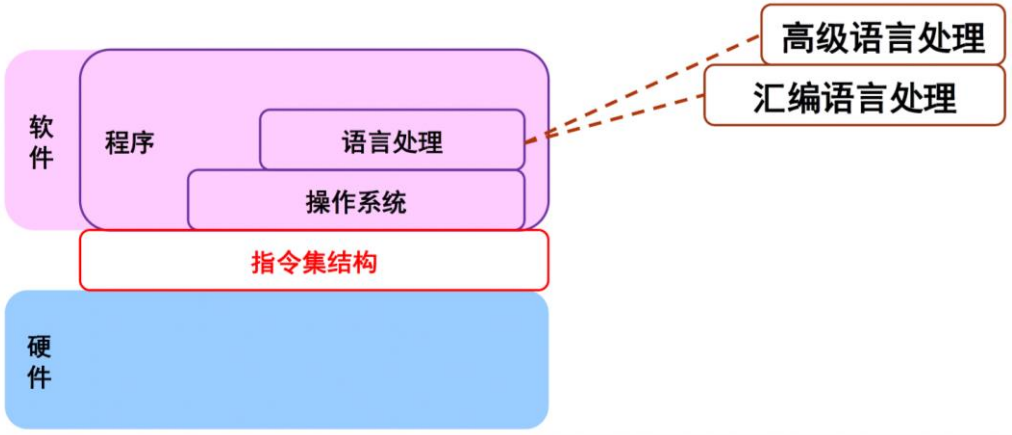
高级语言

- 与底层计算机指令集无关
- “独立于机器”
- 不能直接被计算机执行
- 被翻译为目标机器ISA的二进制指令序列



低级语言

- 与执行程序的计算机指令集紧密相关
- 汇编语言
  - 依据指令集的汇编语言格式编写，需经过语言处理，翻译为机器语言才能在计算机上执行
- 机器语言
  - 依据指令集使用二进制编码，直接在计算机上执行，不需要经过语言处理

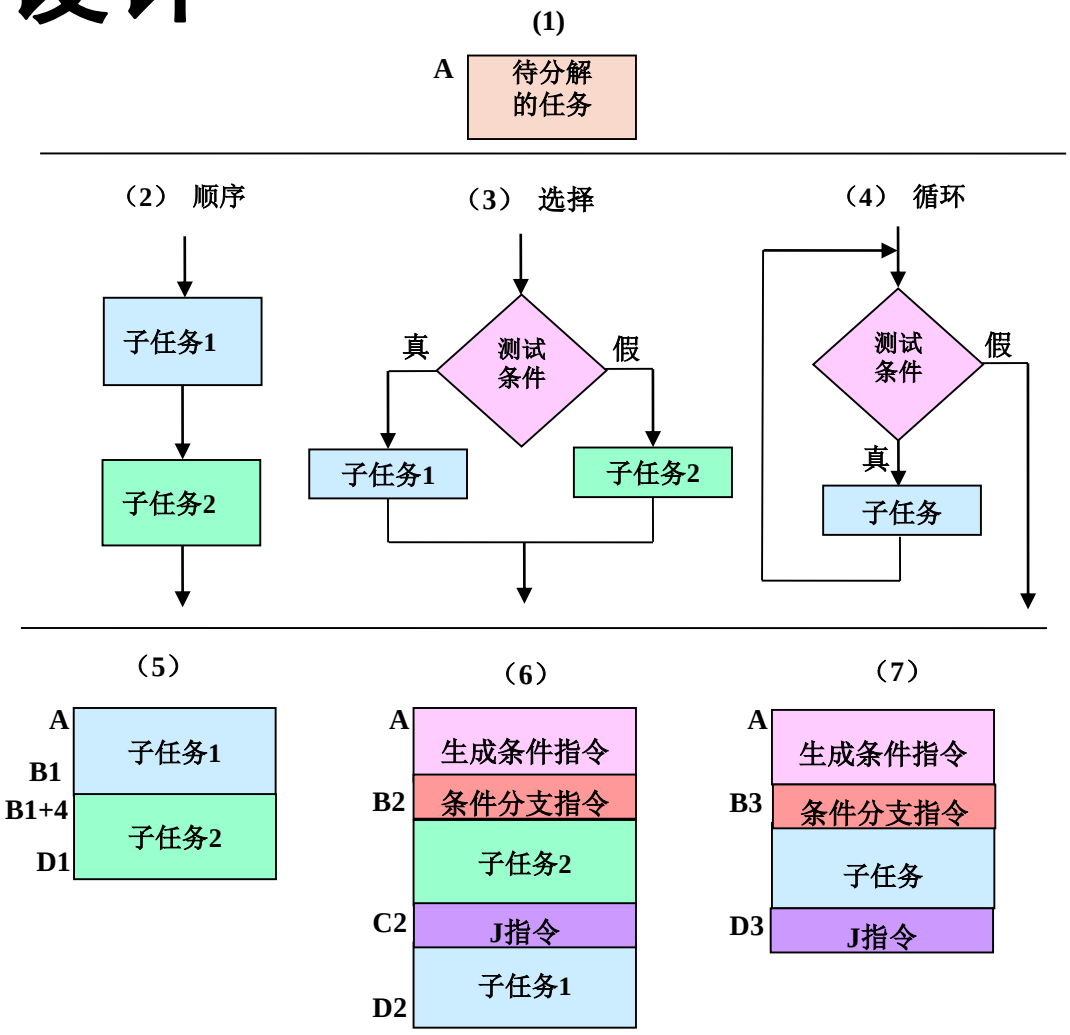


低级语言的作用

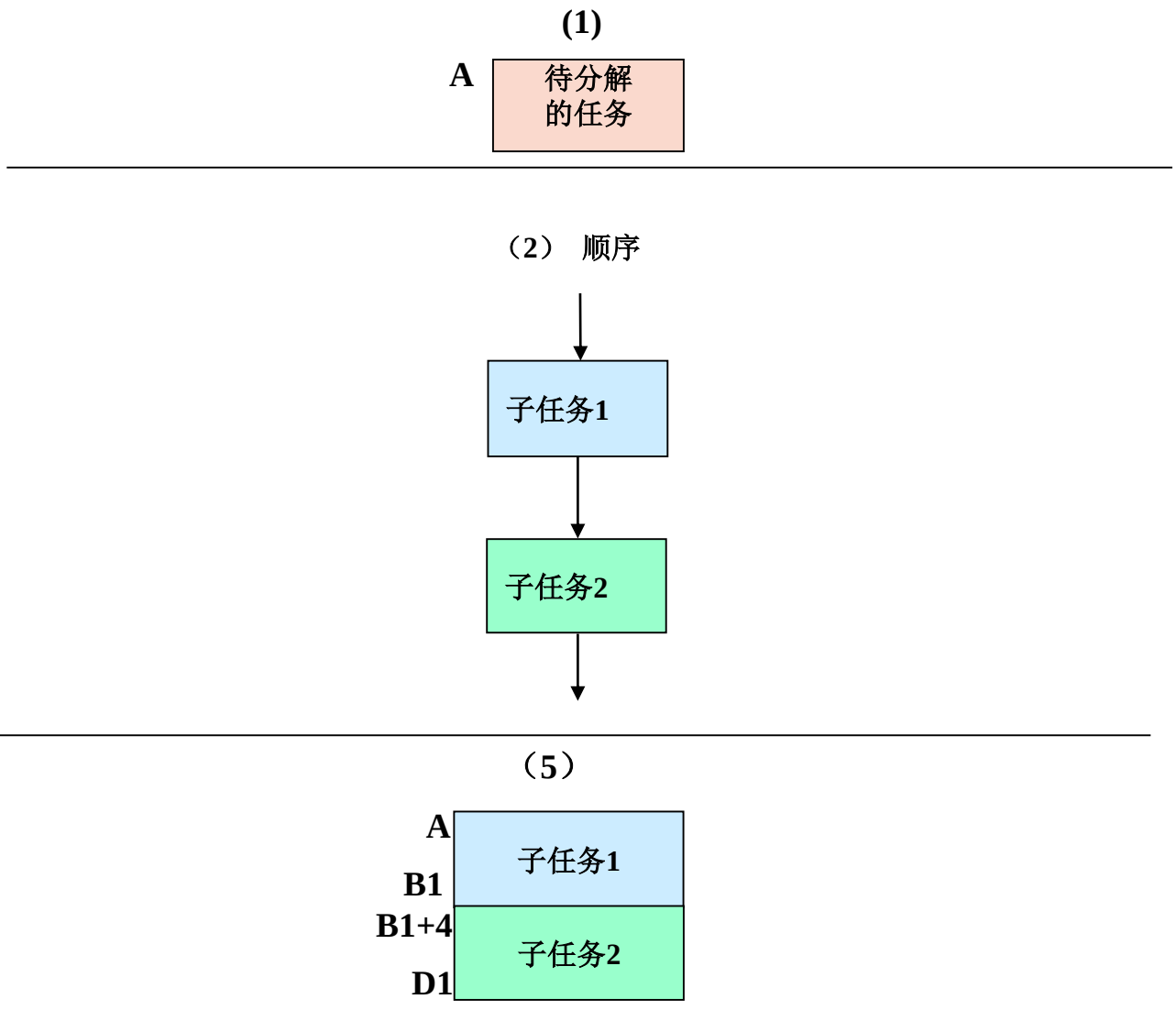
- **硬件控制**：允许程序员直接控制计算机的硬件，如CPU、内存、寄存器等，高度优化的性能关键应用程序和硬件驱动程序。
- **系统编程**：操作系统、设备驱动程序、嵌入式系统等需要与底层硬件交互的领域，需要对计算机硬件有深刻的理解和控制。
- **性能优化**：允许程序员更好地控制寄存器和内存使用，在某些情况下，使用低级语言可以比使用高级编程语言获得更高的性能。
- **调试和逆向工程**：研究恶意软件、漏洞分析以及系统逆向工程等安全领域，允许深入分析和理解二进制代码的内部工作方式。

结构化程序设计

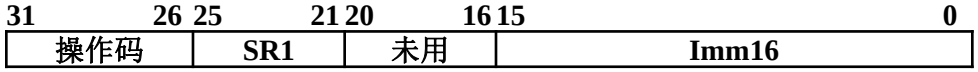
- 三种基本结构
  - 顺序
  - 选择
  - 循环



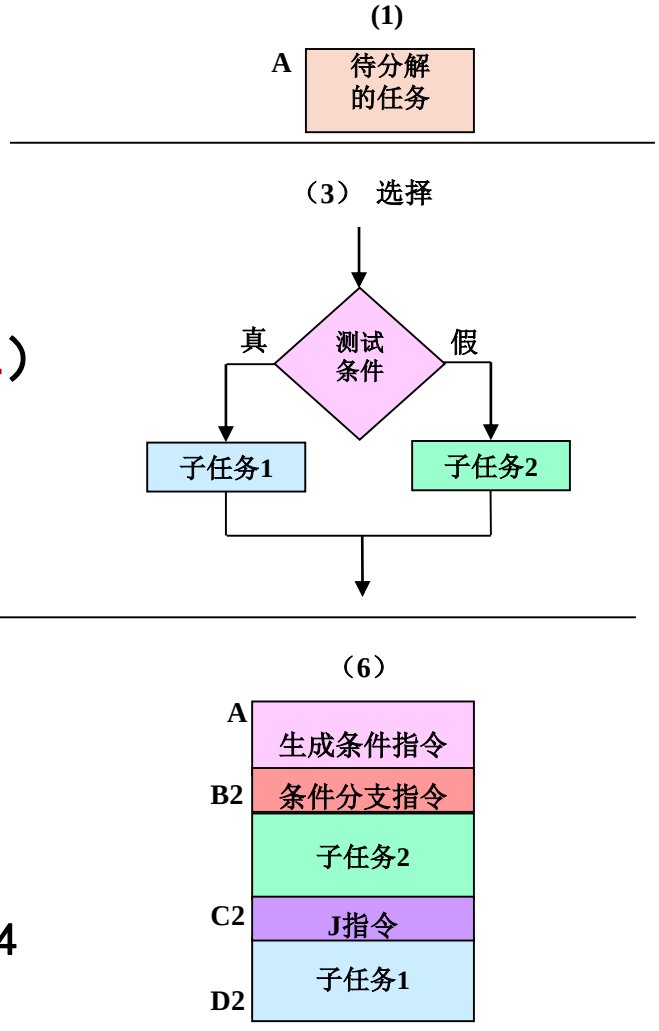
顺序



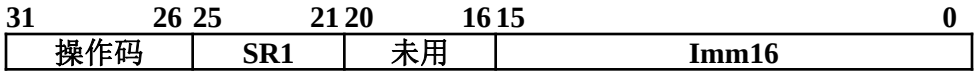
选择



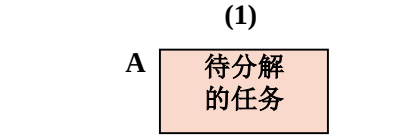
- 一组指令序列生成条件
  - 将某个寄存器Rx设置为零（假）/非零（真）
- 地址B2“条件分支指令”测试该寄存器
  - 条件为真（BNEZ Rx, Y）
    - PC<-C2+4
    - 立即数Y:子任务2的指令数目加1后再乘以4
  - 条件为假
    - PC <- B2+4
    - 子任务2
      - 终止于C2中的无条件跳转指令
      - PC <- D2+4
      - J指令中的立即数：子任务1的指令数目乘以4

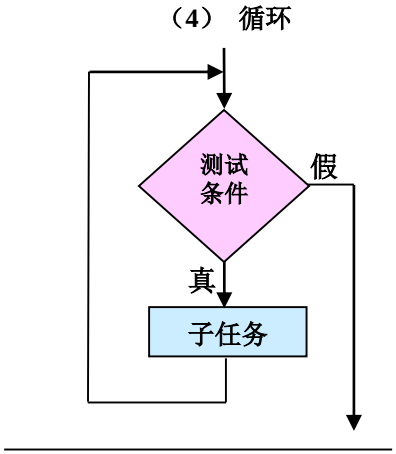


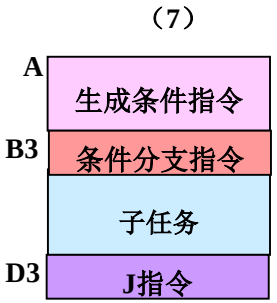
循环



- 一组指令序列生成条件
  - 将某个寄存器Rx设置为零（假）/非零（真）
- 地址B3“条件分支指令”测试该寄存器
  - 条件为假（BEQZ Rx, Y）
    - PC <- D3+4
    - 立即数Y：子任务的指令数目加1后再乘以4
  - 条件为真
    - PC <- B3+4
    - 子任务
      - 结束于D3中的无条件跳转指令
      - PC <- A
    - 问题：J指令中的立即数应为多少？







本章重点

- 课程内容回顾
- 机器语言程序设计
- 示例

示例1：文档加密

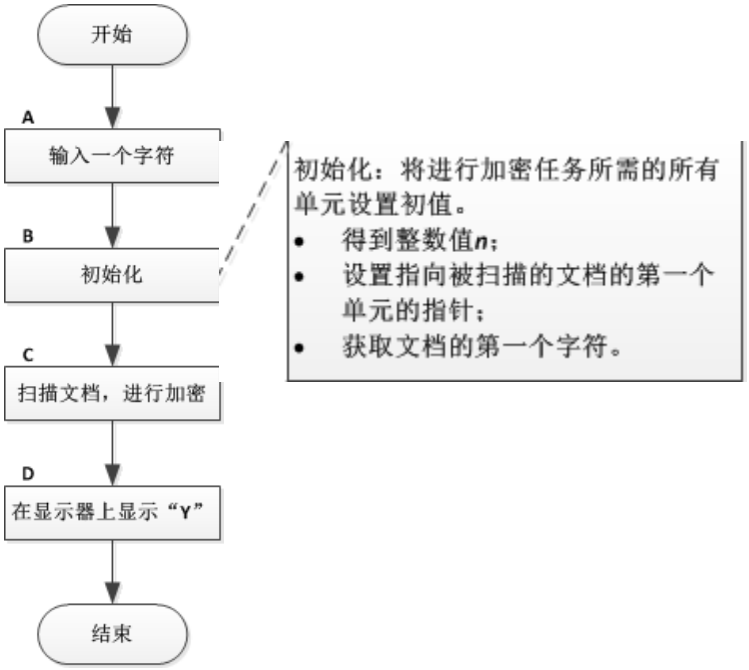
- 根据键盘输入的数值*n*（0到9之间的整数，ASCII码x30~x39），对文档进行加密
- 文档有一个个字符组成，终止标识EOT(x04)
- 加密算法：
  - 如果文档中的字符ASCII码值大于“126-*n*”，将该字符减去“94-*n*”，并替换原来的字符；
  - 而其他字符则加上*n*，进行替换；
  - 最后在显示器上显示字符“Y”(x59)，加密结束。
  - 假设文档中的字符ASCII码值在33~126范围内。

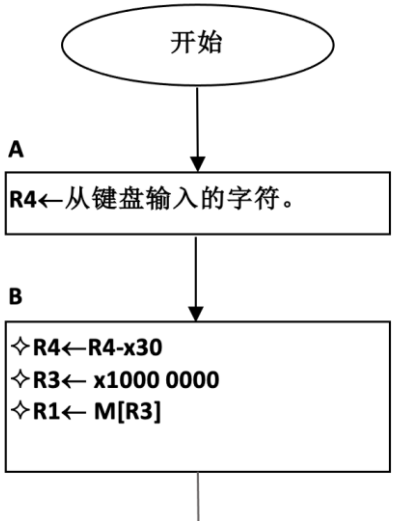
示例1：文档加密

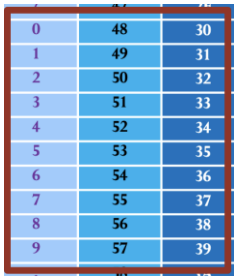
字符	ASCII		字符	ASCII		字符	ASCII		字符	ASCII	
	D	H		D	H		D	H		D	H
NUL	0	00	SP	32	20	@	64	40	`	96	60
SOH	1	01	!	33	21	A	65	41	a	97	61
STX	2	02	"	34	22	B	66	42	b	98	62
ETX	3	03	#	35	23	C	67	43	c	99	63
EOT	4	04	$	36	24	D	68	44	d	100	64
ENQ	5	05	%	37	25	E	69	45	e	101	65
ACK	6	06	&	38	26	F	70	46	f	102	66
BEL	7	07	'	39	27	G	71	47	g	103	67
BS	8	08	(	40	28	H	72	48	h	104	68
HT	9	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
FS	28	1C	<	60	3C	\	92	5C	|	124	7C
GS	29	1D	=	61	3D	]	93	5D	}	125	7D
RS	30	1E	>	62	3E	^	94	5E	~	126	7E
US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

系统分解过程

- 分解为由4个子任务组成的顺序结构
  - 初始化：得到数值*n*，将指针指向被检查文档中第一个字符的地址，然后从被检查文档中提取第一个字符。



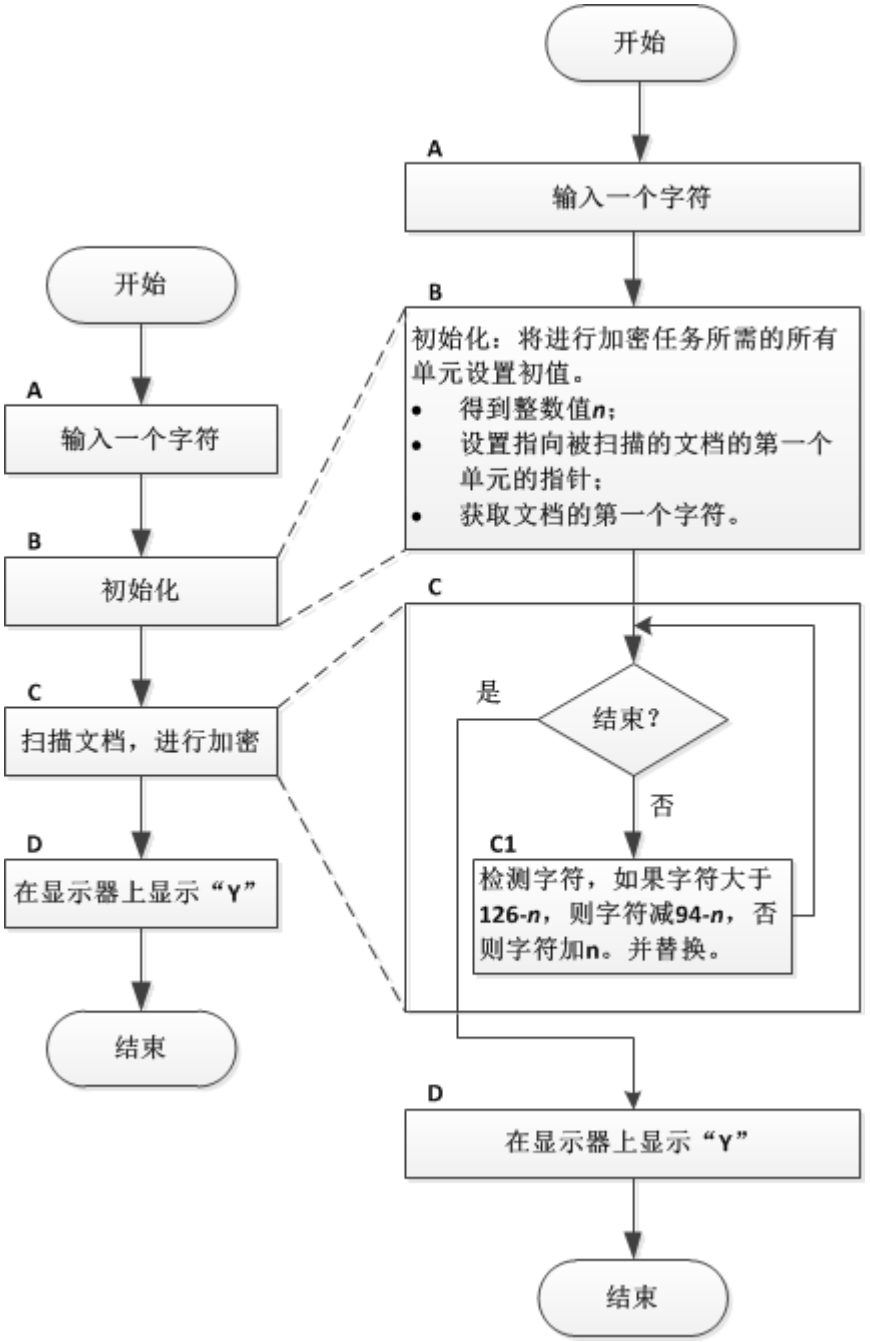




0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39

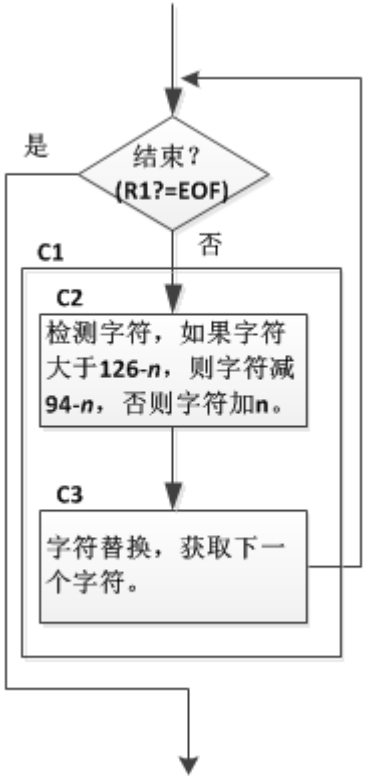
分解C

- 循环结构：只要该文档还有字符需要加密
  - 文档结束，标志为EOT（传输结束，ASCII码为00000100）

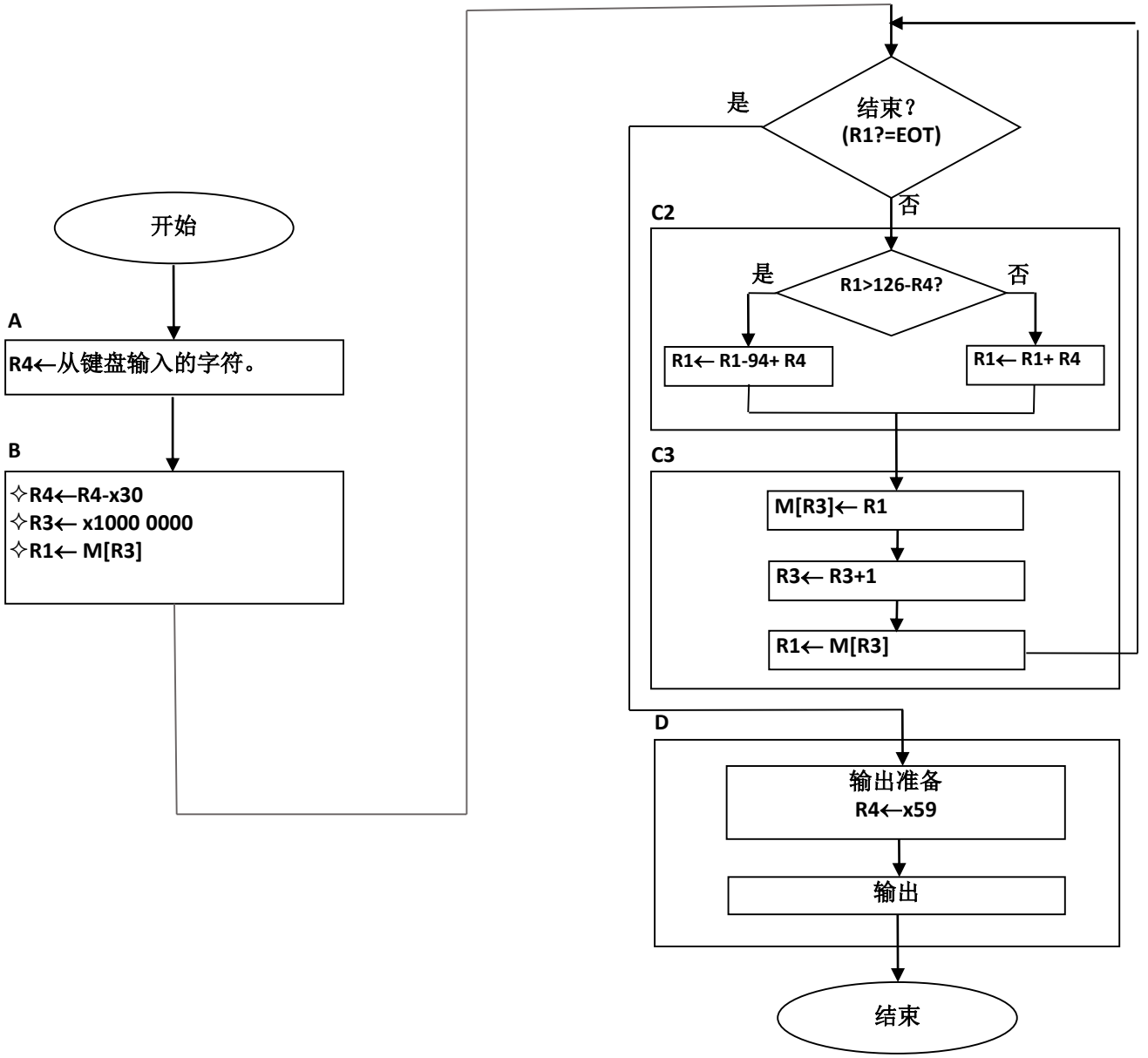


分解C1

- 两个顺序的子任务C2和C3

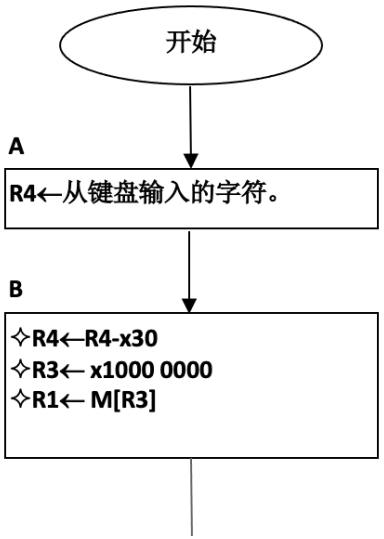


- 使用**选择结构**代替C2
- 用顺序结构实现C3

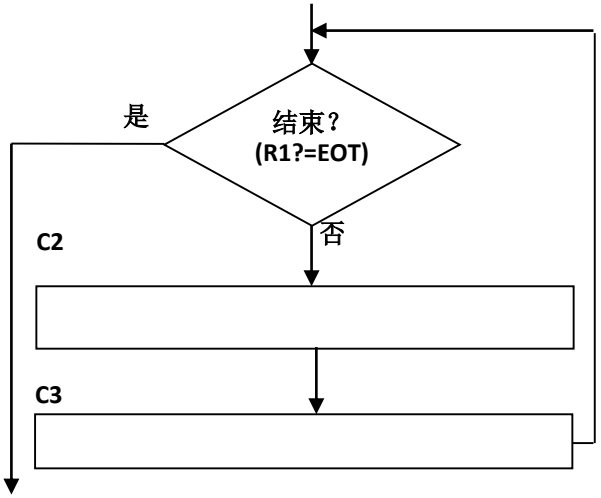


A和B顺序结构

地址	31 26	25 21	20 16	15 11	10 6	5 0	
x0400 0000	110000	000000 0000 0000 0000 0000 0110					TRAP x06/IN
x0400 0004	000011	00100	00100	0000 0000 0011 0000			SUBI R4, R4, x30
x0400 0008	001100	00000	00011	0001 0000 0000 0000			LHI R3, x1000
x0400 000C	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0010	010100	00001	00010	0000 0000 0000 0100			SEQI R2, R1, #4
x0400 0014	101001	00010	00000 0000 0011 0000				BNEZ R2, x30
x0400 0018	000001	00000	00101	0000 0000 0111 1111			ADDI R5, R0, x7F
x0400 001C	000000	00101	00100	00101	000000	000011	SUB R5, R5, R4
x0400 0020	000000	00001	00101	00010	000000	010000	SLT R2, R1, R5
x0400 0024	101001	00010	00000 0000 0000 1100				BNEZ R2, x0C
x0400 0028	000011	00101	00101	0000 0000 0010 0001			SUBI R5, R5, x21
x0400 002C	000000	00001	00101	00001	000000	000011	SUB R1, R1, R5
x0400 0030	101100	000000 0000 0000 0000 0000 0100					J x04
x0400 0034	000000	00001	00100	00001	000000	000001	ADD R1, R1, R4
x0400 0038	010111	00011	00001	0000 0000 0000 0000			SB 0(R3), R1
x0400 003C	000001	00011	00011	0000 0000 0000 0001			ADDI R3, R3, #1
x0400 0040	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0044	101100	111111 1111 1111 1111 1100 1000					J #-56
x0400 0048	000001	00000	00100	0000 0000 0101 1001			ADDI R4, R0, x59
x0400 004C	110000	000000 0000 0000 0000 0000 0111					TRAP x07/OUT
x0400 0050	110000	000000 0000 0000 0000 0000 0000					TRAP x00/HALT

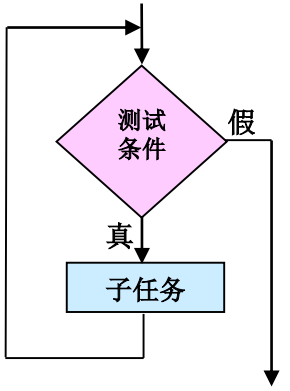


C循环结构

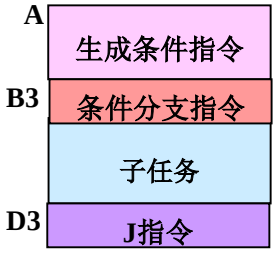


地址	31 26	25 21	20 16	15 11	10 6	5 0	
x0400 0000	110000	000000 0000 0000 0000 0000 0110					TRAP x06/IN
x0400 0004	000011	00100	00100	0000 0000 0011 0000			SUBI R4, R4, x30
x0400 0008	001100	00000	00011	0001 0000 0000 0000			LHI R3, x1000
x0400 000C	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0010	010100	00001	00010	0000 0000 0000 0100			SEQI R2, R1, x04
x0400 0014	101001	00010	00000 0000 0011 0000				BNEZ R2, x30
x0400 0018	000001	00000	00101	0000 0000 0111 1111			ADDI R5, R0, x7F
x0400 001C	000000	00101	00100	00101	000000	000011	SUB R5, R5, R4
x0400 0020	000000	00001	00101	00010	000000	010000	SLT R2, R1, R5
x0400 0024	101001	00010	00000 0000 0000 1100				BNEZ R2, x0C
x0400 0028	000011	00101	00101	0000 0000 0010 0001			SUBI R5, R5, x21
x0400 002C	000000	00001	00101	00001	000000	000011	SUB R1, R1, R5
x0400 0030	101100	000000 0000 0000 0000 0000 0100					J x04
x0400 0034	000000	00001	00100	00001	000000	000001	ADD R1, R1, R4
x0400 0038	010111	00011	00001	0000 0000 0000 0000			SB 0(R3), R1
x0400 003C	000001	00011	00011	0000 0000 0000 0001			ADDI R3, R3, #1
x0400 0040	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0044	101100	111111 1111 1111 1111 1100 1000					J #-56
x0400 0048	000001	00000	00100	0000 0000 0101 1001			ADDI R4, R0, x59
x0400 004C	110000	000000 0000 0000 0000 0000 0111					TRAP x07/OUT
x0400 0050	110000	000000 0000 0000 0000 0000 0000					TRAP x00/HALT

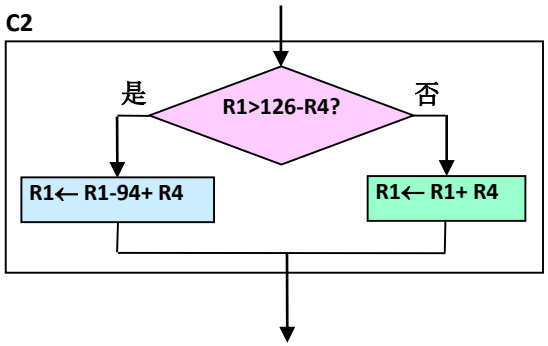
（4） 循环

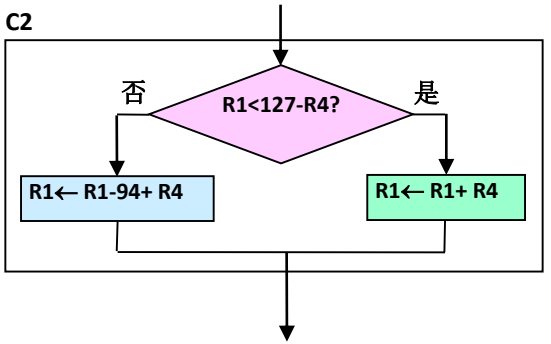


（7）



C2选择结构

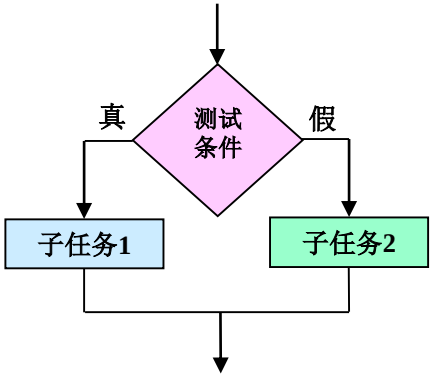




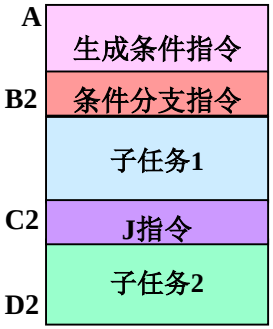
地址	31 26	25 21	20 16	15 11	10 6	5 0	
x0400 0000	110000	000000 0000 0000 0000 0110					TRAP x06/IN
x0400 0004	000011	00100	00100	0000 0000 0011 0000			SUBI R4, R4, x30
x0400 0008	001100	00000	00011	0001 0000 0000 0000			LHI R3, x1000
x0400 000C	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0010	010100	00001	00010	0000 0000 0000 0100			SEQI R2, R1, #4
x0400 0014	101001	00010	00000 0000 0011 0000				BEQZ R2, x30
x0400 0018	000001	00000	00101	0000 0000 0111 1111			ADDI R5, R0, x7F
x0400 001C	000000	00101	00100	00101	000000	000011	SUB R5, R5, R4
x0400 0020	000000	00001	00101	00010	000000	010000	SLT R2, R1, R5
x0400 0024	101001	00010	00000 0000 0000 1100				BNEZ R2, x0C
x0400 0028	000011	00101	00101	0000 0000 0010 0001			SUBI R5, R5, x21
x0400 002C	000000	00001	00101	00001	000000	0 11	SUB R1, R1, R5
x0400 0030	101100	000000 0000 0000 0000 0000 0100					J x04
x0400 0034	000000	00001	00100	00001	000000	00001	ADD R1, R1, R4
x0400 0038	010111	00011	00001	0000 0000 0000 0000			SB 0(R3), R1
x0400 003C	000001	00011	00				DI R3, R3, #1
x0400 0040	010110						0(R3)
x0400 0044	1011						
x0400 0048							0, x59
x0400 004C							T
x0400 0050							LT

R5=127（x7F）
127-R4（n）<=>R5-R4
R1<R5-R4 则R2=1（SLT指令）
R1>=R5-R4 则R2=0

（3） 选择

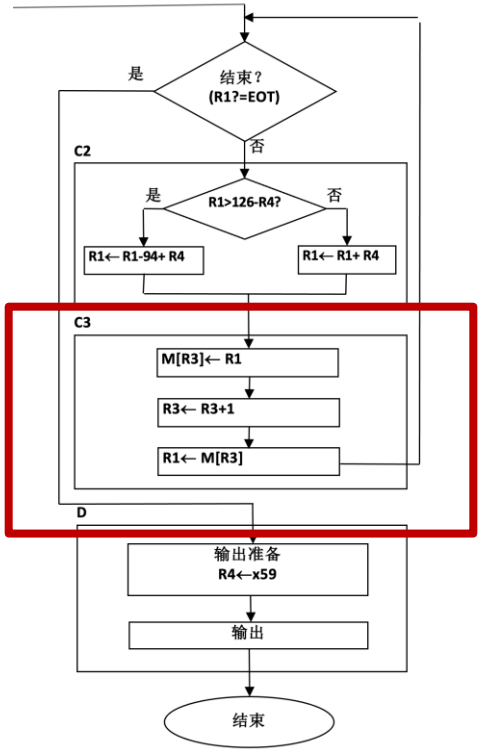


（6）



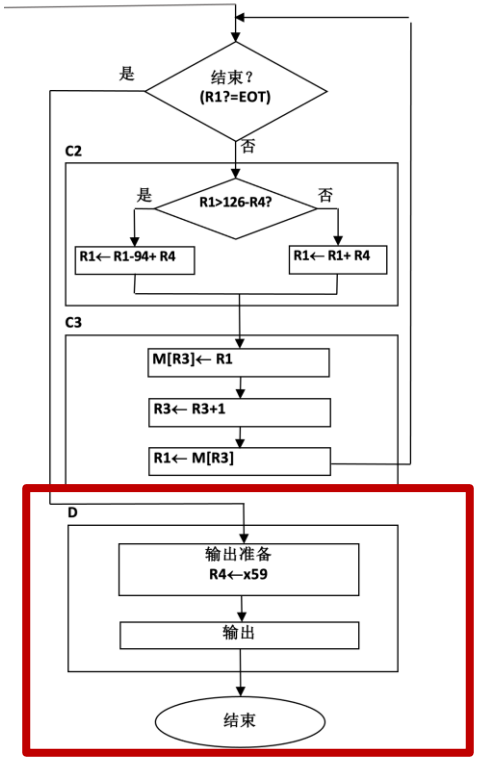
C3顺序结构

地址	31 26	25 21	20 16	15 11	10 6	5 0	
x0400 0000	110000	000000 0000 0000 0000 0000 0110					TRAP x06/IN
x0400 0004	000011	00100	00100	0000 0000 0011 0000			SUBI R4, R4, x30
x0400 0008	001100	00000	00011	0001 0000 0000 0000			LHI R3, x1000
x0400 000C	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0010	010100	00001	00010	0000 0000 0000 0100			SEQI R2, R1, #4
x0400 0014	101001	00010	00000 0000 0011 0000				BEQZ R2, x30
x0400 0018	000001	00000	00101	0000 0000 0111 1111			ADDI R5, R0, x7F
x0400 001C	000000	00101	00100	00101	000000	000011	SUB R5, R5, R4
x0400 0020	000000	00001	00101	00010	000000	010000	SLT R2, R1, R5
x0400 0024	101001	00010	00000 0000 0000 1100				BNEZ R2, x0C
x0400 0028	000011	00101	00101	0000 0000 0010 0001			SUBI R5, R5, x21
x0400 002C	000000	00001	00101	00001	000000	000011	SUB R1, R1, R5
x0400 0030	101100	000000 0000 0000 0000 0000 0100					J x04
x0400 0034	000000	00001	00100	00001	000000	000001	ADD R1, R1, R4
x0400 0038	010111	00011	00001	0000 0000 0000 0000			SB 0(R3), R1
x0400 003C	000001	00011	00011	0000 0000 0000 0001			ADDI R3, R3, #1
x0400 0040	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0044	101100	111111 1111 1111 1111 1100 1000					J #-56
x0400 0048	000001	00000	00100	0000 0000 0101 1001			ADDI R4, R0, x59
x0400 004C	110000	000000 0000 0000 0000 0000 0111					TRAP x07/OUT
x0400 0050	110000	000000 0000 0000 0000 0000 0000					TRAP x00/HALT



D顺序结构

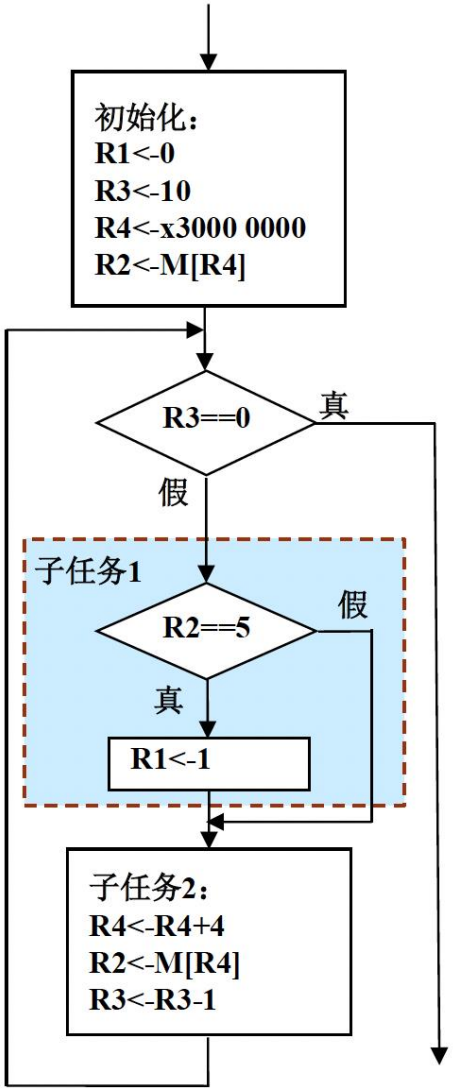
地址	31 26	25 21	20 16	15 11	10 6	5 0	
x0400 0000	110000	000000 0000 0000 0000 0000 0110					TRAP x06/IN
x0400 0004	000011	00100	00100	0000 0000 0011 0000			SUBI R4, R4, x30
x0400 0008	001100	00000	00011	0001 0000 0000 0000			LHI R3, x1000
x0400 000C	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0010	010100	00001	00010	0000 0000 0000 0100			SEQI R2, R1, #4
x0400 0014	101001	00010	00000 0000 0011 0000				BEQZ R2, x30
x0400 0018	000001	00000	00101	0000 0000 0111 1111			ADDI R5, R0, x7F
x0400 001C	000000	00101	00100	00101	000000	000011	SUB R5, R5, R4
x0400 0020	000000	00001	00101	00010	000000	010000	SLT R2, R1, R5
x0400 0024	101001	00010	00000 0000 0000 1100				BNEZ R2, x0C
x0400 0028	000011	00101	00101	0000 0000 0010 0001			SUBI R5, R5, x21
x0400 002C	000000	00001	00101	00001	000000	000011	SUB R1, R1, R5
x0400 0030	101100	000000 0000 0000 0000 0000 0100					J x04
x0400 0034	000000	00001	00100	00001	000000	000001	ADD R1, R1, R4
x0400 0038	010111	00011	00001	0000 0000 0000 0000			SB 0(R3), R1
x0400 003C	000001	00011	00011	0000 0000 0000 0001			ADDI R3, R3, #1
x0400 0040	010110	00011	00001	0000 0000 0000 0000			LB R1, 0(R3)
x0400 0044	101100	111111 1111 1111 1111 1100 1000					J #-56
x0400 0048	000001	00000	00100	0000 0000 0101 1001			ADDI R4, R0, x59
x0400 004C	110000	000000 0000 0000 0000 0000 0111					TRAP x07/OUT
x0400 0050	110000	000000 0000 0000 0000 0000 0000					TRAP x00/HALT



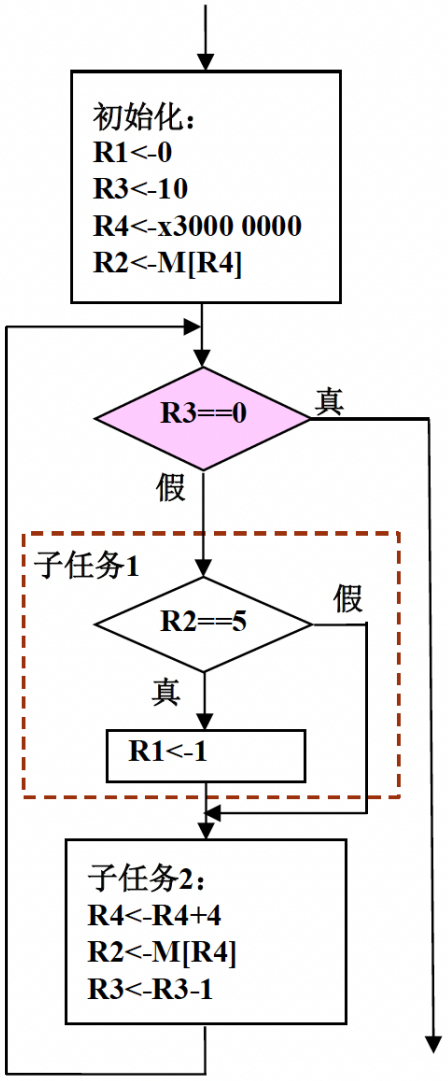
示例2：判断连续存储单元内是否包含5

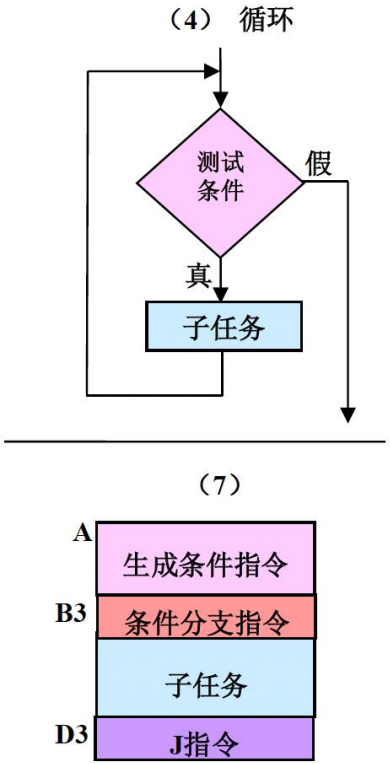
- 检查：
- 从地址x3000 0000开始存储的10个整数
  - 有5，R1设置为1
  - 没有5，R1为0

- 计数器控制的循环
  - R3，计数器
- 子任务1
  - 选择结构



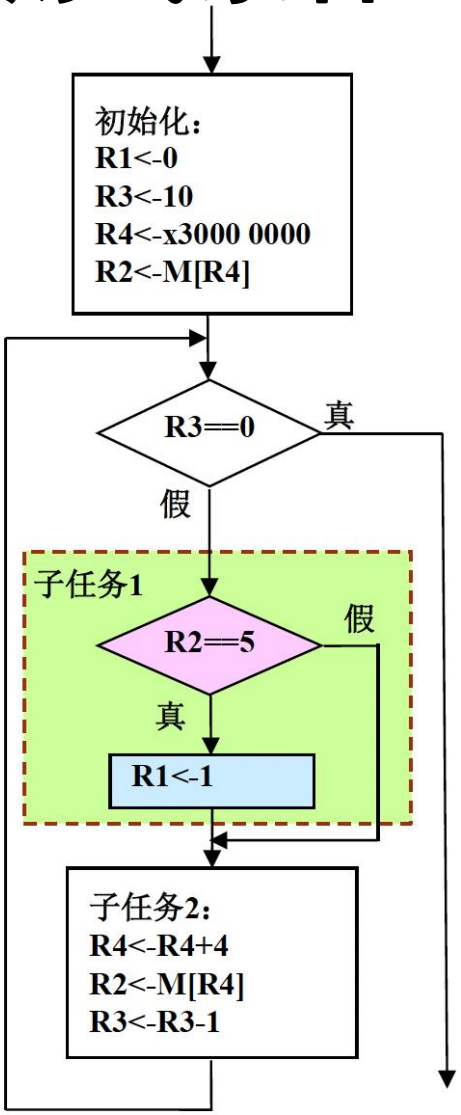
测试条件 R3==0

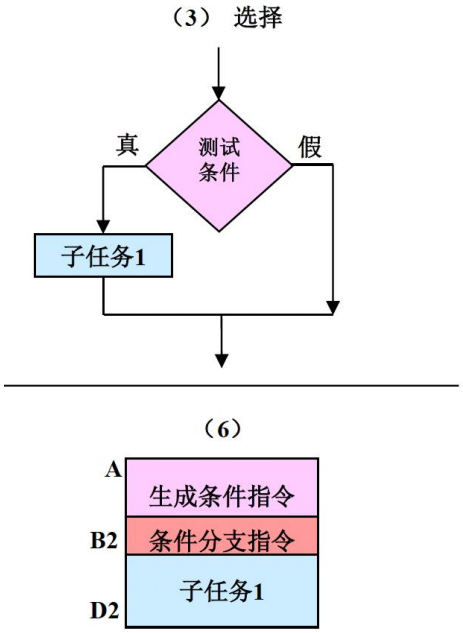




- 不需要生成条件指令
- 条件分支指令
  - BEQZ R3, D3+4

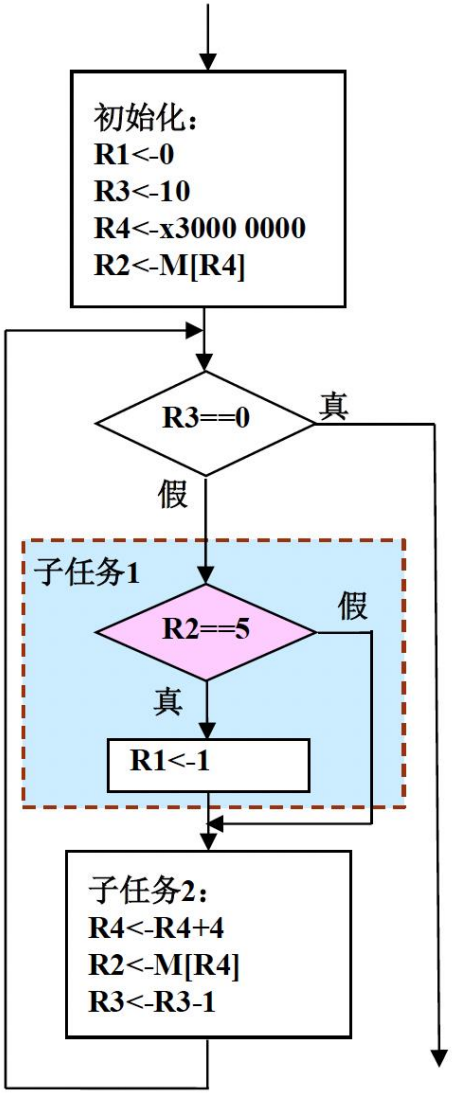
测试条件 R2==5





- 生成条件指令
  - SEQI Rx, R2, #5
- 条件分支指令
  - BEQZ Rx, D2+4

机器语言程序



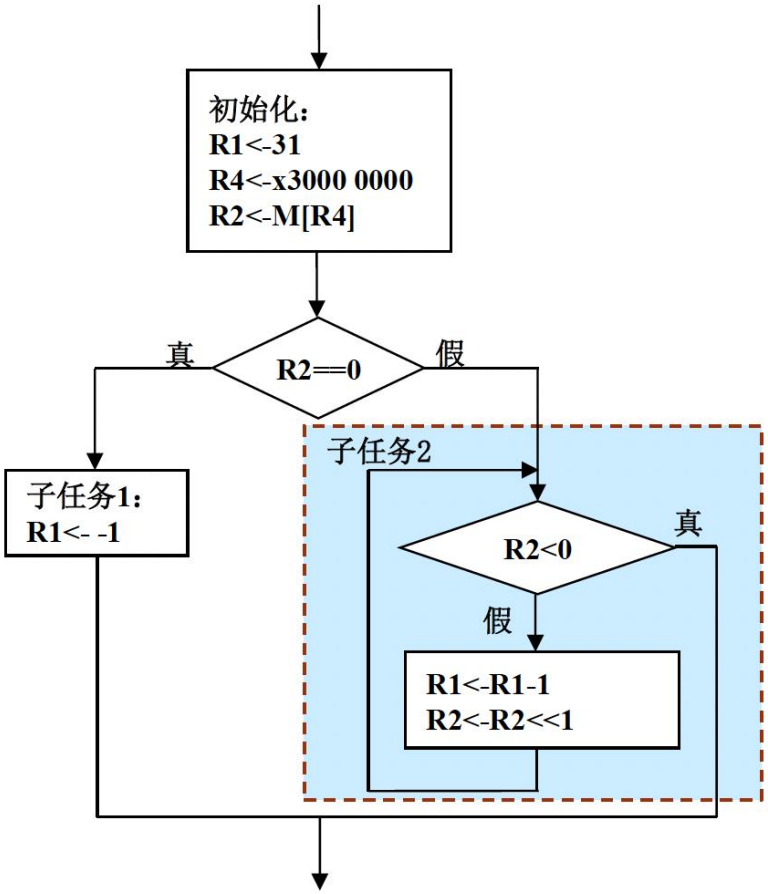
31 26	25 21	20 16	15 11	10 6	5 0	解释
001001	00001	00001	0000 0000 0000 0000			ANDI R1,R1, #0
000001	00000	00011	0000 0000 0000 1010			ADDI R3,R0, #10
001100	00000	00100	0011 0000 0000 0000			LHI R4, x3000
011100	00100	00010	0000 0000 0000 0000			LW R2, 0(R4)
101000	00011	00000	0000 0000 0010 0000			BEQZ R3, #32
010100	00010	00101	0000 0000 0000 0101			SEQI R5, R2, #5
101000	00101	00000	0000 0000 0000 1000			BEQZ R5, #8
000001	00000	00001	0000 0000 0000 0001			ADDI R1,R0, #1
101100	00 0000 0000 0000 0000 0001 0000					J #16
000001	00100	00100	0000 0000 0000 0100			ADDI R4,R4, #4
011100	00100	00010	0000 0000 0000 0000			LW R2, 0(R4)
000011	00011	00011	0000 0000 0000 0001			SUBI R3,R3, #1
101100	11 1111 1111 1111 1111 1101 1100					J #-36
……						

- 选择结构
  - 当R2为5时，设置R1为1
  - 使用J指令跳出循环

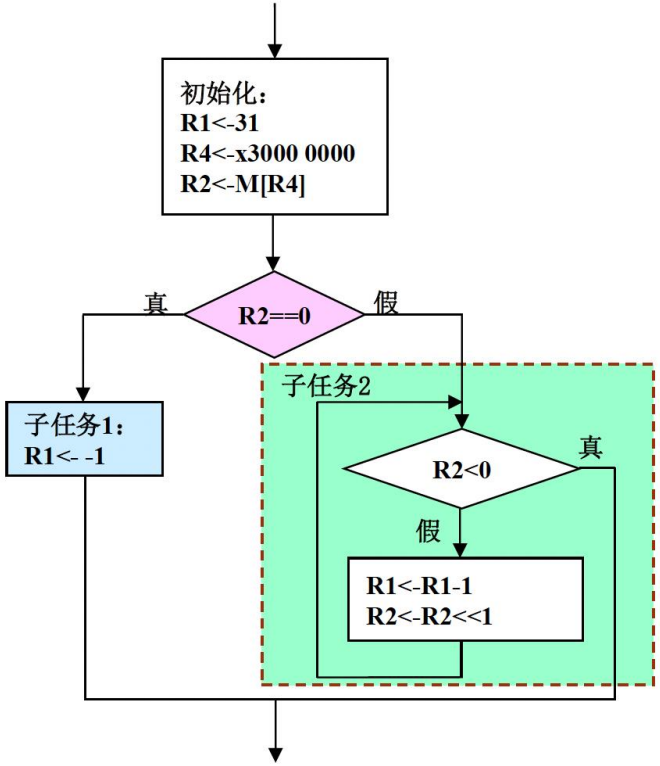
示例3：找到字中的第一个“1”

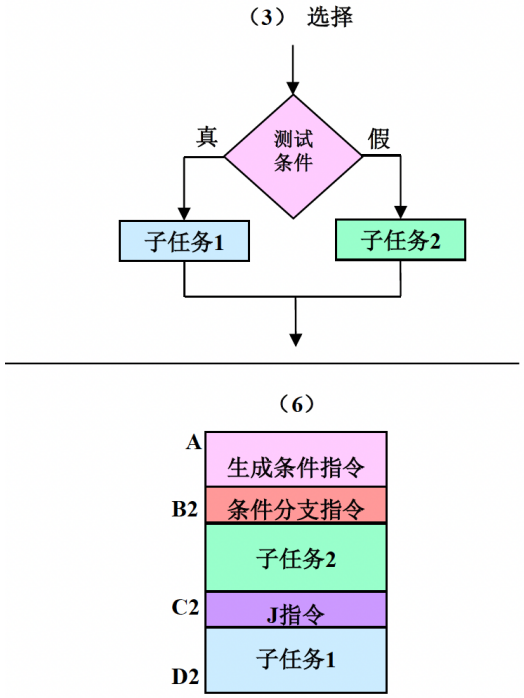
- 检查：
  - x3000 0000~x3000 0003中的字
  - 找出第一个“1”（从左到右）
    - 存储到R1中
  - 如果没有1
    - R1 <- -1
  - 例如
    - 0010 0000 0000 0000 0000 0000 0000 0000, R1=29
    - 0000 0000 0000 0000 0000 0000 0010 0000, R1=5

- 选择结构
- 子任务2
  - 标志控制的循环
  - 标志
    - R2<0：R2[31]=1
  - 循环子任务
    - R2=R2<<1
      - R2[30]，R2[29]…==1？



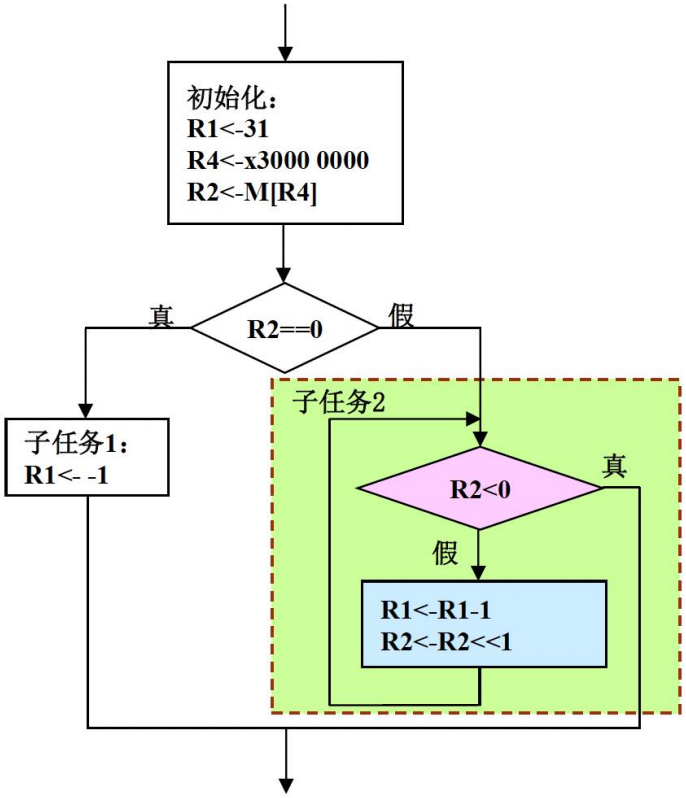
测试条件 R2==0

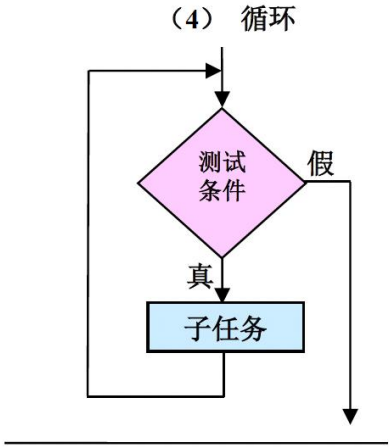


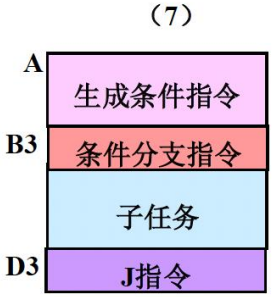


- 不需要生成条件指令
- 条件分支指令
  - BEQZ R2, C2+4

测试条件 R2<0



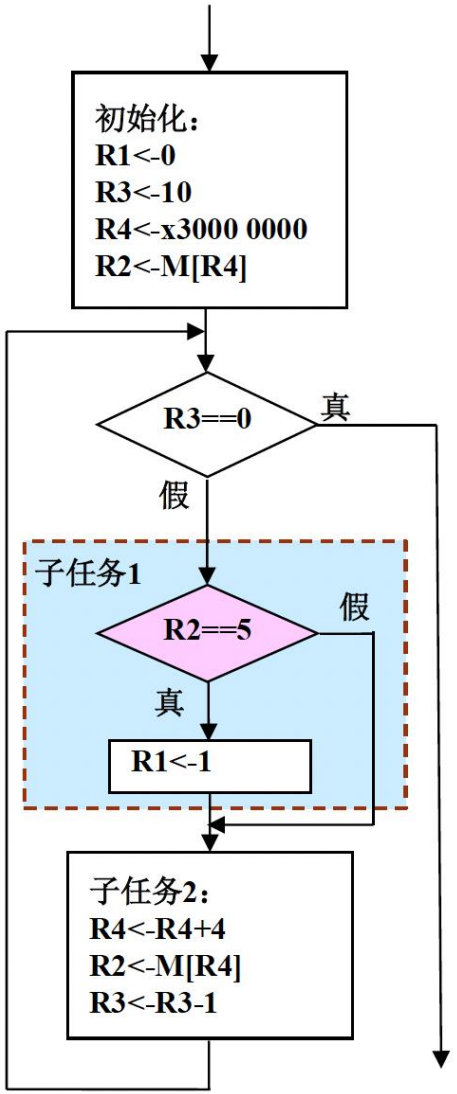




- 生成条件指令
  - SLTI Rx, R2, #0
- 条件分支指令
  - BNEZ Rx, D3+4

机器语言程序

31 26	25 21	20 16	15 11	10 6	5 0	解释
000001	00000	00001	0000 0000 0001 1111			ADDI R1,R0, #31
001100	00000	00100	0011 0000 0000 0000			LHI R4, x3000
011100	00100	00010	0000 0000 0000 0000			LW R2, 0(R4)
101000	00010	00000	0000 0000 0001 0100			BEQZ R2,#20
010000	00010	00011	0000 0000 0000 0000			SLTI R3, R2, #0
101001	00011	00000	0000 0000 0001 0000			BNEZ R3, #16
000011	00001	00001	0000 0000 0000 0001			SUBI R1,R1, #1
001101	00010	00010	0000 0000 0000 0001			SLLI R2,R2, #1
101100	111111 1111 1111 1111 1110 1100					J #-20
001010	00000	00001	1111 1111 1111 1111			ORI R1,R0, #-1
……						



书面作业

- 10. 1
- 10. 2