

7.2 操作数类型和操作种类

一、操作数类型

- 地址 无符号整数
- 数字 定点数、浮点数、十进制数
- 字符 ASCII
- 逻辑数 逻辑运算

0				
4				
8				

二、数据在存储器中的存放方式

例 1 2 3 4 5 6 7 8 H 的存放方式

0	12H	34H	56H	78H
4				
8				

字地址 为 高字节 地址

0	78H	56H	34H	12H
4				
8				

字地址 为 低字节 地址

二、数据在存储器中的存放方式

— 字节编址，数据在存储器中的存放方式（存储字长**64**位，机器字长**32**位）

a. 从任意位置开始存储

a. 从任意位置开始

... xx00	字 节	半 字		双-----			
... xx08	字		单 字				半---
... xx10	字	单 字			字节	单---	
... xx18	字						
... xx20							

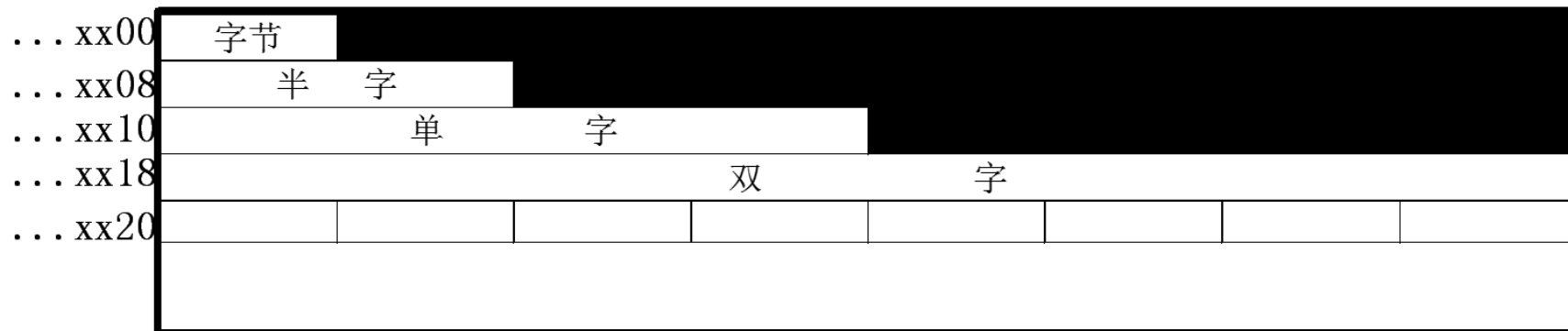
优点：不浪费存储资源

缺点：除了访问一个字节之外，访问其它任何类型的数据，都可能花费两个存储周期的时间。读写控制比较复杂。

二、数据在存储器中的存放方式

— 字节编址—数据在存储器中的存放方式

b. 从一个存储字的起始位置开始访问



优点： 无论访问何种类型的数据，在一个周期内均可完成，读写控制简单。

缺点： 浪费了宝贵的存储资源

二、数据在存储器中的存放方式

— 字节编址—数据在存储器中的存放方式

c.边界对准方式——从地址的整数倍位置开始访问

(c) 边界对准方式—按地址的整数倍存储

...xx00	字节	浪 费					
...xx08	双 字						
...xx10	半 字	浪 费					
...xx18	双 字						
...xx20	单 字			浪 费			
...xx28	双 字						
...xx30	字节	浪 费			单 字		
...xx38	半 字	浪 费		单 字			
...xx40	字节	浪费	半 字				

数据存放的起始地址是数据长度（按照编址单位进行计算）的整数倍

本方案是前两个方案的折衷，在一个周期内可以完成存储访问，空间浪费也不太严重。

三、操作类型

7.2

1. 数据传送

源	寄存器	寄存器	存储器	存储器
目的	寄存器	存储器	寄存器	存储器
例如	MOVE	STORE MOVE PUSH	LOAD MOVE POP	MOVE
置“1”，清“0”				

2. 算术逻辑操作

加、减、乘、除、增 1、减 1、求补、浮点运算、十进制运算
与、或、非、异或、位操作、位测试、位清除、位求反

如 8086 **ADD SUB MUL DIV INC DEC CMP NEG**
 AAA AAS AAM AAD
 AND OR NOT XOR TEST

3. 移位操作

算术移位 逻辑移位

循环移位（带进位和不带进位）

4. 转移

(1) 无条件转移 **JMP**

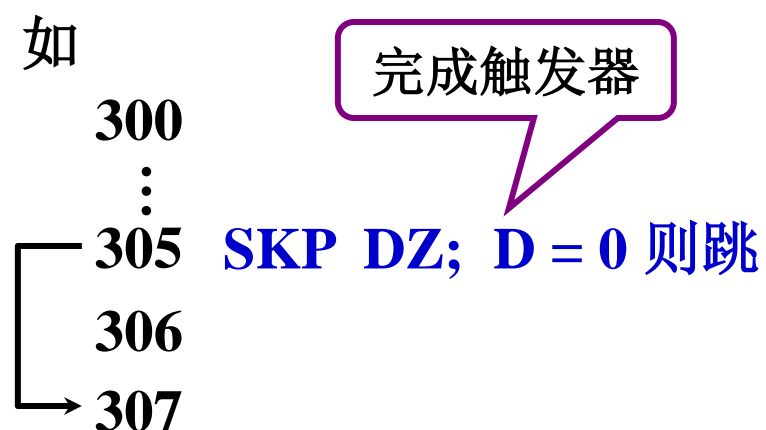
(2) 条件转移

结果为零转 ($Z = 1$) **JZ** 如

结果溢出转 ($O = 1$) **JO** 300

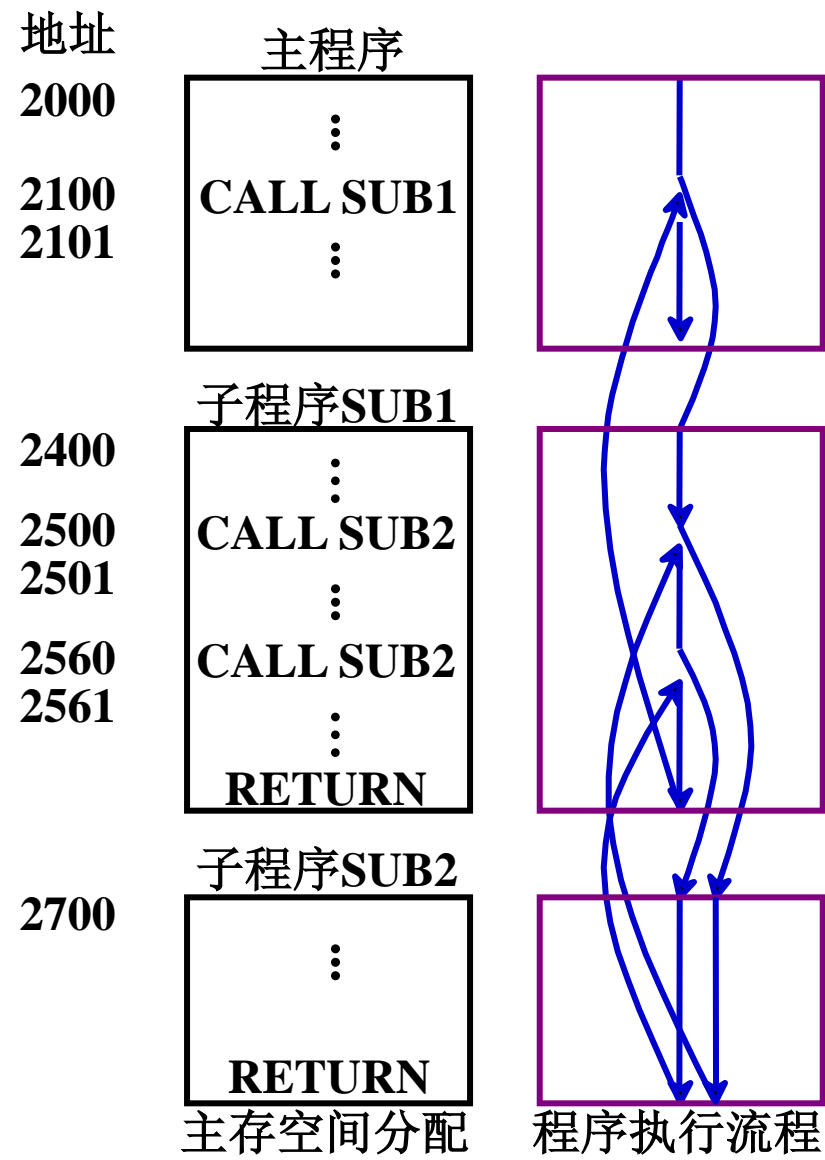
结果有进位转 ($C = 1$) **JC** \vdots

跳过一条指令 **SKP**



(3) 调用和返回

7.2



(4) 陷阱 (Trap) 与陷阱指令

7.2

意外事故的中断

- 一般不提供给用户直接使用

在出现事故时, 由 CPU 自动产生并执行 (隐指令)

- 设置供用户使用的陷阱指令

如 8086 **INT TYPE** 软中断

提供给用户使用的陷阱指令, 完成系统调用

5. 输入输出

入 端口中的内容 \longrightarrow CPU 的寄存器

如 **IN AK, n** **IN AK, DX**

出 CPU 的寄存器 \longrightarrow 端口中的内容

如 **OUT n, AK** **OUT DX, AK**