#### 体系往年大题合集，雄起！！！

### 99年

三、计算分析题（45分）

1. （12分）对于条件转移指令，我们有一下两种实现方案：

CPU1：用2条指令完成（1条比较指令和1条转移指令）

CPU2：用1条指令完成（1条比较且转移指令）

假设对于两个CPU而言，条件转移指令都需要2个时钟周期，所有其它指令都仅需要1个时钟周期。而对CPU1，20%的执行指令是条件转移指令，因此还有20%是比较指令。由于CPU1在转移指令中没有比较功能，因此其时钟周期比CPU2快10%。试比较这两个CPU的性能。

（2分）

（3分）

（2分）

（3分）

（2分）因此是CPU2性能好。

2．（13分）已知Cache的命中时间为1个时钟周期。当Cache失配时，存储器访问延时为6个时钟周期，传输速率为每个时钟周期传输1个字（4 Bytes）。已知一2路组关联、回写、写分配Cache，块大小为32 Bytes。有一写缓冲器，数据送入写缓冲器需额外花一个时钟周期。假定写缓冲器的命中率为70%。经Cache模拟器测试，得到以下实验数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 总次数 | 取指访存次数 | 数据访存次数 | 读操作数访存次数 | 写操作数访存次数 |
| 存储访问 | 4000 | 1880 | 2120 | 936 | 1184 |
| 访存失配 | 192 | 80 | 112 | 64 | 48 |

从存储器取入Cache的存储字：768字

写入Cache的总字数： 168字

试计算平均存储访问时间（要求写出计算分析的思路）。

（4分）写分配：（80+64+48）\*（6+1\*8）=192\*14=2688

（2分）回写次数：168/4=42

（4分）回写代价：42\*(70%\*1 + (1－70%)\*(6+1\*8))=205.8

（3分）1.723

3.（20分）分析下列代码序列在基本整数流水线上执行情况：（假定R3的初始值是R2+400，采用的是数据和指令分离Cache，且所有存储器访问均命中Cache，转移目标地址计算和条件判断在ID级进行）

LOOP: LW R2, 0(R1)

SUB R1, R1, #4

LW R3, 0(R1)

ADD R3, R2, R3

SW 0(R1), R3

BNEZ R1, LOOP

(6分)如果流水线寄存器结构竞争时，仅采用先写后读（即在同一时钟周期内可同时进行一个寄存器读和一个寄存器写操作，前半拍写，后半拍读），不采用任何旁路机构。画出指令序列执行的流水状态图（一次迭代即可）。假定转移处理采用冲刷流水线策略。

(6分)假定整数流水线使用旁路硬件结构，用同样方式画出上述代码序列执行的流水状态图（一次迭代即可），假定转移处理采用预测转移不发生方案。

(8分)假定流水线的转移延时槽是单个时钟周期，且具有标准硬件旁路结构。可以调度指令进入转移延时槽。请你重新编译新指令序列，以便取得最佳性能，可修改个别指令的操作数，但不能进行循环变换改变循环体内指令的操作码数目。画出新代码序列执行的流水状态图（一次迭代即可），并计算执行整个循环需要多少时钟周期。

（1）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **流水时钟** | | | | | | | | | | | | | | |
| LW | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |  |
| SUB |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |
| LW |  |  | IF | S | S | ID | EX | DM | WB |  |  |  |  |  |  |
| ADD |  |  |  |  |  | IF | S | S | ID | EX | DM | WB |  |  |  |
| SW |  |  |  |  |  |  |  |  | IF | S | S | ID | EX | DM | WB |
| BNEZ |  |  |  |  |  |  |  |  |  |  |  | IF | ID | EX | DM |
|  |  |  |  |  |  |  |  |  |  |  |  |  | IF | IF |  |

（2）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **流水时钟** | | | | | | | | | | | | | | |
| LW | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |  |
| SUB |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |
| LW |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |
| ADD |  |  |  | IF | ID | S | EX | DM | WB |  |  |  |  |  |  |
| SW |  |  |  |  | IF | S | ID | EX | DM | WB |  |  |  |  |  |
| BNEZ |  |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |
| 转移 |  |  |  |  |  |  |  | IF | Idle | Idle |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | IF |  |  |  |  |  |  |

(3a) Loop: LW R2, 0(R1)

LW R3, -4(R1)

SUB R1, R1, #4

ADD R3, R2, R3

BNEZ R1, LOOP

SW 0(R1), R3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **流水时钟** | | | | | | | | | | | | | | |
| LW | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |  |
| LW |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |
| SUB |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |
| ADD |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |
| BNEZ |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |
| SW |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |
|  |  |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |

Time = 6 \* 99 + 10 = 604

(3b) LW R2, 0(R1)

Loop: LW R3, -4(R1)

SUB R1, R1, #4

ADD R3, R2, R3

SW 0(R1), R3

BNEZ R1, LOOP

LW R2, 0(R1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **流水时钟** | | | | | | | | | | | | | | |
| LW | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |  |
| LW |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |  |
| SUB |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |  |
| ADD |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |  |
| SW |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |  |
| BNEZ |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |  |
| LW |  |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |  |
|  |  |  |  |  |  |  |  | IF | ID | EX | DM | WB |  |  |  |

Time = 1 + 6\*99 + 10 = 605

评分建议：

流水线状态图每个6分，其中第3小题如果指令序列合理，可以给3分。

计算时钟周期2分。

### 00年

1. (8) Assume the unpipelined processor has a 1ns clock cycle and that it uses four cycles for ALU operations and branches and five cycles for memory operations. Assume that the relative frequencies of these operations are 50%, 20%, and 30%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?

2. (8) Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 60% of the time, measured as a percentage of the execution time when the enhanced mode is in use. What percentage of the original execution time has been converted to fast mode?

3. (10 ) Let’s use an in-order execution computer, such as the UltraSPARC III. Assume the cache miss penalty is 100 clock cycles, and all instructions normally take 1.0 clock cycles (ignoring memory stalls). Assume the average miss rate is 2%, there is an average of 1.5 memory references per instruction, and that the average number of cache misses per 1000 instructions is 30.

1. Calculate the average memory access time.
2. Calculate the impact on performance using **BOTH** misses per instruction and miss rate.
3. How is the performance improved when behavior of the cache is included?

4. (16 ) Consider the 5-stage pipeline that we have considered in class. This implementation has a double-pumped register file , independent ports for instruction and memory accessing, and ALL possible forwarding paths. We’ll execute the following code segment on this pipeline. (You can assume that R4 has its value in the register file at the start of the program execution):

lw R1, 0(R4) # (1)

addi R2, R1, 15 # (2)

addi R3, R1, 10 # (3)

xor R1, R1, R2 # (4)

or R2, R3, R2 # (5)

add R3, R1, R2 # (6)

sw R3, 1(R4) # (7)

1. Please draw out the pipeline execution of the above instructions in the table below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Lw R1,0(R4) | IF | ID | EX | Mem | WB |  |  |  |  |  |  |  |  |  |  |
| addi R2, R1, 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| addi R3, R1, 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Xor R1, R1, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| or R2, R3, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add R3, R1, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sw R3, 1(R4) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Lw R1,0(R4) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| addi R2, R1, 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| addi R3, R1, 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Xor R1, R1, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| or R2, R3, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add R3, R1, R2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sw R3, 1(R4) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

(2) Describe all the forwarding paths in the 5-stage implementation that would help it run THIS segment of code better. For each forwarding path, list the stage that it would originate from (SOURCE), the stage that it would terminate in (SINK), and the instructions in the code segment that require that path. (There are more slots below than you need.)

|  |  |  |
| --- | --- | --- |
| Source | SINK | INSTRUCIONS |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

(3) Could the compiler do anything to help the 5-stage implementation (with no forwarding paths) execute this piece of code more efficiently? If so, please briefly describe what it can do. If not, please briefly describe why not.

### 02年

1. 在采用一位转移预测缓冲器结构中，分析计算计算机执行下列程序时，其转移的成功概率和最差情况硬件的预测命中率是多少？

**for(i=0；i<=3；i++)**

**for(j=0；j<100；j++)**

**a[i]+=b[i];**

1. 在某主存/Cache层次结构中，1个Block由2个字构成。Cache由SRAM芯片承担，其响应时间是8ns。主存贮器由DRAM承担，其响应时间是70ns。现有下列C语言片段，设数组中的每个元素是1个字，除数组外，其它变量均在寄存器中，请分析该C语言片段执行时，最坏情况时数据Cache的失配率是多少？平均存储器访问时间是多少？

**for ( i=0； i<100； i++)**

**s=s+A[i] ;**

1. 设指令间的相关性参数如下表，试分析计算下列问题：
   1. 计算该循环迭代一次需多少时钟。
   2. 采用软件流水方式编译优化下列循环，使其循环内的竞争最少。
   3. 计算优化后该循环迭代一次需多少时钟。

|  |  |  |
| --- | --- | --- |
| 前操作指令 | 后继相关指令 | 延迟时钟 |
| FP ALU 操作 | FP ALU 操作 | 3 |
| FP ALU 操作 | Store (双字) | 2 |
| Load (双字) | FP ALU 操作 | 1 |
| Load (双字) | Store (双字) | 0 |

**LOOP： L.D F0，0(R1)**

**ADD.D F4，F0，F2**

**S.D F4，0(R1)**

**DAAUI R1，R1，#-8**

**BNE R1，R2，LOOP**

1. 设总线上有10个处理器企图同时锁定一个变量。每个总线事务（读失配和写失配）需100个时钟，忽略保存在Cache内锁的读、写时间及获得锁的时间。假设在时刻**t0**锁被释放，且所有处理器都在自旋，又假设所有处理器速度相同，总线是公平的，对所有请求服务顺序响应。试计算确定：
   1. 分析获得10个锁的过程。
   2. 10个处理器均获得锁所需的总路线事务个数。
   3. 处理10个处理器请求约需多少时间。

答案

* 1. 设：预测缓冲器初值B=0

预测策略是： B=1，预测转移成功

B=0，预测转移不成功

程序的转移行为：“i”循环4次，其中3次转移成功（i=0,1,2）,1次转移不成功（i=3）

“j”循环4×100，其中4×99转移成功（j=0~98）,4×1次不成功

**程序中转移指令的成功概率是：399/404=98.76%**

预测器的行为：初始B=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 内循环 | | | 外循环 | |
| j=0 | j=1~98 | j=99 |  | |
| 失配B=0→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=0 | 失配B=0→B=1 |
| 命中B=1→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=1 | 失配B=0→B=1 |
| 命中B=1→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=2 | 失配B=0→B=1 |
| 命中B=1→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=3 | 命中B=0→B=0 |

**硬件的预测命中率是：(404-8)/404=396/404=98.02%**

**另解**

设：预测缓冲器容量>=2，预测缓冲器初值 B=0，B1=0

预测策略是： B=1，B1=1预测转移成功

B=0，B1=0预测转移不成功

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 内循环 | | | 外循环 | |
| j=0 | j=1~98 | j=99 |  | |
| 失配B=0→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=0 | 失配B1=0→B1=1 |
| 失配B=0→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=1 | 命中B1=1→B1=1 |
| 失配B=0→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=2 | 命中B1=1→B=1 |
| 失配B=0→B=1 | 命中B=1→B=1 | 失配B=1→B=0 | i=3 | 失配B1=1→B1=0 |

**硬件的预测命中率是：(404-10)/404=394/404=97.5%**

* 1. 设：最坏的情况为初始整个数组均不在Cache内

程序在读数组A[i]时有50次读失配：即读A[0]、A[2]、……A[98]

此50读失配时，Cache读入50个块，将A[1]、A[3]……A[99]也读入Cache内

因此，数组A[i]读偶数元素时为命中，故数据Cacahe失配率为：

Miss rate = 50/100 = 50%

存储器平均访问时间：AMT ＝ 8×50% + 70×50% ＝39

* 1. （1）

LOOP： L.D F0，0(R1) 1

Stall 2

ADD.D F4，F0，F2 3

Stall、stall 4~5

该循环执行一次迭代需10个时钟

S.D F4，0(R1) 6

DAAUI R1，R1，#-8 7

Stall 8

BNE R1，R2，LOOP 9

Stall 10

（2） L.D F0，0(R1) 1

DAAUI R1，R1，#-8 2

ADD.D F4，F0，F2 3

L.D F0，0(R1) 4

DAAUI R1，R1，#-8 5

LOOP：S.D F4，*16*(R1) 1

ADD.D F4，F0，F2 2

L.D F0，0(R1) 3

BNE R1，R2，LOOP 4

DAAUI R1，R1，#-8 5

ADD.D F4，F0，F2 1

S.D F4，*16*(R1) 2

Stall 3

S.D F4，*8*(R1) 4

* 1. 优化后该循环执行一次迭代需 5＋9/n 个时钟

**（1）**设：某处理器释放锁值后，该锁值为该处理器独占，此时i 个处理器同时竞争锁值，只有一个处理器能获得锁值，因此需：

**i** 次读锁值（load linked）操作 ——i 个处理器同时读失配

**i** 次锁定锁值（store conditional）操作——1个处理器写无效，i－1个处理器写失配

**1**次释放锁值 ——该处理器释放锁值，写无效，独占锁值

此时，i－1个处理器又开始同时竞争锁值，重复上述过程，直到i=0为止。

对n 个处理器，均获取一次锁的总路线事务数是：

**n**

**i=1**

**∑（2i＋1）＝n2＋２＝102＋2×10＝120**

* 1. 处理10个处理器请求时间约需：

120×100＝12000

### 03年

**四、下面是一段不常规代码。（10分）**

For (I = 1; I<100; I = I + 1) {

A[ I ] = B[ I ] + C[ I ]; /\* S1 \*/

B[ I ] = A [ I ] + D[ I ]; /\* S2 \*/

A[ I + 1 ] = A[ I ] + E[ I ]; /\* S3 \*/

**}**

1. 请分析指令间的相关性（输出相关、反相关、真相关），并且注明该相关是否循环传递（Loop-carried）。
2. 该循环是否可以并行？如果不是，请重写该循环使之成为并行的。

**五、某集中共享存储器多处理器系统（SMP），基于写时无效（write invalidate）的侦听（snooping）协议，支持Cache一致性（coherence）。请填写下表完成Cache数据块的状态转换。（15分）**

|  |  |  |  |
| --- | --- | --- | --- |
| 请求信息 | 信息来源 | Cache当前状态 | Cache下一状态 |
| 读命中 | 处理器 | 共享（share） |  |
| 读失配 | 处理器 | 共享（share） |  |
| 写命中 | 处理器 | 共享（share） |  |
| 写失配 | 处理器 | 共享（share） |  |
| 读失配 | 总线 | 共享（share） |  |
| 写失配 | 总线 | 共享（share） |  |
| 读失配 | 总线 | 占有（exclusive） |  |
| 读命中 | 处理器 | 占有（exclusive） |  |
| 读失配 | 处理器 | 占有（exclusive） |  |
| 写命中 | 处理器 | 占有（exclusive） |  |
| 写失配 | 处理器 | 占有（exclusive） |  |
| 写失配 | 总线 | 占有（exclusive） |  |
| 读失配 | 处理器 | 无效（invalid） |  |
| 写失配 | 处理器 | 无效（invalid） |  |

**六、某分布式共享存储器多处理器系统（DSM），基于写时无效（write invalidat）的目录（Directory-Based），支持Cache一致性（coherence）。填写下表完成目录中数据块的状态变化。share是拥有对应数据块的处理器的集合。P是发出请求的处理器（7分）**



|  |  |
| --- | --- |
| ① |  |
| ② |  |
| ③ |  |
| ④ |  |
| ⑤ |  |
| ⑥ |  |
| ⑦ |  |

**七、删栏同步是多处理器并行的典型应用之一，以下是两个实现删栏同步的代码段。**

**（其中**：**TOTAL是总的进程数）（8分）**

***Program1:***

**lock ( lock1 );**

**if (arrives = 0 ) lock2 = 0;**

**arrives = arrives + 1;**

**unlock( lock1 );**

**if (arrives = = TOTAL )**

**arrives = 0;**

**lock2 = 1;**

**}**

**else {**

**spin(lock2 = = 1);**

**}**

***Program2:***

**}**

**token =! token;**

**lock( lock1 );**

**arrives = arrives + 1;**

**if ( arrives = = TOTAL ){**

**arrives = 0;**

**lock2 = token;**

**}**

**unlock( lock1 );**

**spin(lock2 = = token);**

**}**

1. 请问Program1能否正确实现删栏同步，为什么？
2. 请问Program1能否正确实现删栏同步，简述token的作用。

**八、在某一多处理器系统中，处理器P1和P2上各有一个程序运行（如下图），其中X和Y都是共享变量，假设初始时X = Y = OLD，X\_copy = Y\_copy = NULL。（10分）**

**P1：**

**Y = NEW;**

**X\_copy = X；**

**L1：**

**P2：**

**X = NEW;**

**Y\_copy = Y;**

**L2：**

1. 请问：在SC（Sequential Consistency）存储器连贯性模型下， X\_copy和Y\_copy的值共有多少种可能性，分别是多少？
2. 请问：在TSO（Total Store Order）存储器连贯性模型下， X\_copy和Y\_copy的值共有多少种可能性，分别是多少？

**九、简述带硬件投机机制的Tomasulo算法中的重构缓存器ROB（reorder buffer）与普通的Tomasulo算法中的保留站（reservation stations）的功能区别。（10分）**

**备注：**上面的第八题由于难度比较大（答案不唯一，有好几种，），所以在实际考试时并没有考。（该题取之美国麻省理工学院，所以在进行考博复习时要特别重视）

### 04年

1. (10)Your company is developing a program with high requirement on computation. You asked your R&D department to make some improvements on the execution time. After several months, they give you two solutions. The first one is to use a new hardware technology, by which 40% of the computation can be accelerated by 10 times. Another solution is focused on algorithm design, which can enhance 60% and 10% of the total computation by 2 and 20 times respectively.

Question:

a) What is the overall enhancement of the hardware solution?

b) What is the overall enhancement of the software solution?

c) Which one will you choose?

Answer:

a):



b):



c): Software solution is better.

2. (13)Within some **memory/cache** memory hierarchies, there are 2 words in a block. Access time form Cache is 8ns and for main memory miss penalty is 70ns. For the code of C language below, assumes that each element is one word in array (**A[i])**. Except array, another variables has be loaded to registers. While the C codes execute, please calculate and questions below:

**for ( i=0; i<100; i++)**

**s=s+A[i] ;**

* 1. What is the miss rate for data accesses?
  2. What is the average memory access time for data read?
  3. What is the overall CPI including memory access? Assume processor runs at 1.1GHz and has a CPI of 1 excluding memory accesses. Ignores instructions misses and data hazard and control hazard. Assumes assembler code is below:

………………

LOOP: LOAD R2, 0(R1)

ADD R5,R1,#4

ADD R3, R2,R3 ;s was stored in R3

BNE R5, LOOP

…………….

**Answer:**

**assumed condition:**

**Block 1 word/block**

**Access time of cache(hit time) 8ns**

**Access time of memory (miss penalty) 70ns**

**CPU clock rate 1GHz**

**Ideal CPI 1**

**All memory accesses 100**

**Clocks for one accesses time/T=time×f=70ns×1.1GHz=77**

1. **For data accesses**

**Misses Accesses for even elements: A[0],A[2],…………………**

**There are 50 misses accesses**

**Miss rate for data is 50/100=50%**

1. **Average memory accesses time**

**AMAT＝hit time×(1-miss rate) ＋ miss rate×miss penalty**

**＝8×(1-50%) ＋50%×70**

**＝39ns**

1. **Overall CPI**

**Number of instructions are 400.**

**CPI＝ideal CPI ＋ (Misses/instructions) × miss penalty**

**＝1＋50/400 × 77**

**＝9.625**

3. (13) Consider the following pipeline. All instructions have five cycles but autoincrement addressing instruction, which is IF, ID, EX, MEM, WB. Branch will complete at the third cycle. The pipeline extended MIPS pipeline in autoincrement addressing mode which have seven pipe stages. The Fig 1 is an example in autoincrement addressing:

eg: Add R1, (R2)+

means: Regs[R2]🡨Regs[R2]+4

Regs[R1]🡨Regs[R1] + Mem[Regs[R2]]

Fig 1: example instruction of autoincrement addressing

The register files can perform two reads and two write every clock cycle. To handle reads and writes to the same register, assume the register write in the first half of the clock cycle and the read in the second half.



IF: Instruction fetch

ID: Instruction decode

ADDR: AutoIncrement Addressing

WB1: Write Result of ADDR to Register file

EX: Memory Reference: Calculate the absolute address

ALU Instruction: Calculate.

MEM: Memory Access

WB2: Write Result to Register file

Fig 2: seven pipeline stage of the autoincrement addressing

Read the following code segment . The pipeline has forwarding path.

Loop: LW R1, 4(R2)

ADD R2, (R1)+

ADDI R3, R3, #4

SUB R4, R1, R2

SW R2, 4(R4)+

BNEZ R3, Loop

(question1)if the pipeline has no delay slot, find out how forwarding path work in every instruction? Draw the pipe stage diagram, mark the every forwarding path in diagram.

(question2)if the pipeline has one delay slot, how to adjust the code segment. Draw the pipe stage diagram.

**答案1：**



评分标准：6分，每条指令0.5分，3个箭头各1分

答案2：

ADDI R3, R3, #4

Loop: LW R1, 4(R2)

ADD R2, (R1)+

SUB R4, R1, R2

SW R2, 4(R4)+

BNEZ R3, Loop

ADDI R3, R3, #4 延时槽

ADDI R3, R3, #-4



评分标准：延时槽内指令1分，循环前后预处理与后处理语句各0.5分。流水线状态图2分

如果有学生画成流水线时空图，看答案是否正确给分。

### 05年 2005冬体系卷子回忆，没有选择题部分的

简答题 4题X6分

1, 用LL，SC实现EXCH R2, 0(R1)

2, Trace Scheduling的目标，主要思想和步骤

3,软件流水的主要思想，给一个例子说明

4，硬件投机的原理，ROB的作用

分析计算 46分

1，相关性分析（真相关，输出相关或反相关），指出是否loop-carried

for(i=0;i<=100;i=i+1)

{

A[i]=B[i]+C[i];

B[i]=A[i]+D[i];

A[i+1]=A[i]+E[i];

}

2，课本上的Example，英文书343页

3，多发射＋投机， 英文书235页的Example，大致类似有些改变

给出指令序列，画出双发射带动态投机的情况下，指令的执行情况，就是填充图3.34这样的图。

4，目录协议，当前状态如下：

p1 TAG Value State

A Invalid

p2 TAG Value State

A Invalid

p3 TAG Value State

A 2 Exclusive

目录中

TAG State Sharers

A Invalid {P3}

给出P1执行x=3的具体过程，指出收发的消息，Sharers的变化，节点上Value的变化