Computer Architecture 3220104111 吴浩无

量化计算公式

① $CPUTime = \frac{InstructionCount \times CPI}{ClockRate}$

② Amdahl's law: $T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$

③ $Sp_{overall} = \frac{T_{old}}{T_{new}} = \frac{1}{(1-f) + \frac{f}{Sp_{optimized}}}$ f←被优化部分所占运行时间比例

④吞吐率 Throughput = $\frac{指令数量}{执行时间}$

⑤加速比 Speedup = $\frac{n \times m}{m+n-1}$ n-指令数量

⑥效率 Efficiency = $\frac{n}{m+n-1}$ m-流水线段数 又称流水线深度

⑦内存性能衡量

⑦-1 从CPU的执行和stall角度分类讨论: Inst access + Data access

$CPUtime = IC \times (CPI_{execute} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty) \times CycleTime$

⑦-2 从CPU的指令类型(ALU指令还是Memory指令)分类讨论

$AMAT = \frac{Whole\ access\ time}{All\ memory\ accesses\ in\ program}$

$= (HitTime_{inst} + MissRate_{inst} \times MissPenalty_{inst}) \times Inst\%$

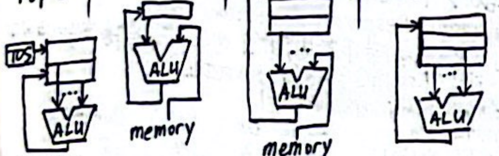
$+ (HitTime_{data} + MissRate_{DATA} \times MissPenalty_{DATA}) \times Data\%$

$CPUtime = (AluOps \times CPI_{AluOps} + MemAccess \times AMAT) \times Cycletime$

ISA boundary between software and hardware

GPR classification: 以A+B为例

Stack	Accumulator	Register RM register-memory	Register RR load-store
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3



设计原则:

①Simplicy favors regularity 规范性 ②Smaller is faster

③make the common case fast ④Good design demands good compromises 妥协、

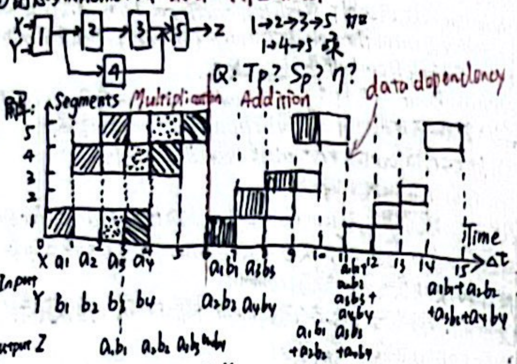
Pipeline

Instruction Dependences	Pipeline Hazards
- Data Dependences → RAW	- Data Hazards
- Name Dependences	- RAW
- Anti-dependence → WAR	- WAR
- Output-dependence → WAW	- WAW
- Control Dependences → Branch Hazards	- Branch Hazards
	- Structural Hazards

流水线计算问题

1. 向量A(a1, a2, a3, a4), 向量B(b1, b2, b3, b4), 算A·B.

①静态多功能流水线 乘法→排空→加法



$\Rightarrow T_p = \frac{7}{15\Delta t}$, $S_p = \frac{12+8+4}{15} = 1.6$, $\eta = 32\%$.

②动态多功能流水线

$\Rightarrow T_p = \frac{7}{18\Delta t}$, $S = \frac{28\Delta t}{18\Delta t} \approx 1.56$, $E = \frac{4\times4+3\times4}{5\times18} \approx 31\%$

2.非线性流水线调度

① Initial conflict vector

k \ n	1	2	3	4	5	6	7	8	9
1	✓								✓
2		✓	✓					✓	
3				✓					
4					✓	✓			
5							✓	✓	

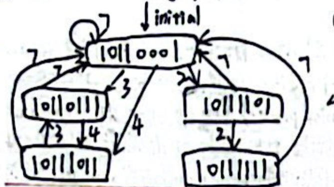
1000 0000
0011 0001
0
0000 0001
0000 0001
(取或)
⇒ 10110001

② Conflict vector

Interval	Initial	2	2	7
CCV	10110001	10111101	10111111	10110001
1→	10110001	00101100	00001011	00000000
2→		10110001	00101100	00000000
3→			10110001	00000001
4→				10110001

找到一个循环调度: 2-2-7

③ state transition graph



Circular queue:

2,2,7	2,7	3,4
4,3	3,4,7	3,7
4,3,7	4,7	7

Block Identification

Block Address		offset
Tag	Index	

Tag: 用于判断是否存在某个block stored in cache and used in comparison with CPU address

Index: selects set

offset: selects data within the block

内存性能衡量问题

1. Clock Rate = 200MHz, Ideal CPI = 1.1
50% arith/logic, 30% ld/st, 20% control
10% of memory operations get 50 cycle miss penalty
1% of instructions → same

Q: CPU time and the AMAT?

A: CPI = ideal CPI + average stalls per instruction

$= 1.1 + \underbrace{0.3 \times 0.1 \times 50}_{data\ miss} + \underbrace{1 \times 0.01 \times 50}_{inst\ miss} = 3.1$

$AMAT = \frac{1}{1.3} \times (1 + 0.01 \times 50) + \frac{0.3}{1.3} \times (1 + 0.1 \times 50) = 2.54$ cycles

(inst; hit cycle; miss cycle; data)

2. Ideal CPI = 1, 50% data access, miss penalty: 25 cycle
Miss rate: 2% Q: How faster if all inst cache hits?

A: Memory stall cycles = $IC \times \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty$

$= IC \times (1+0.5) \times 0.02 \times 25 = IC \times 0.75$.

$\frac{IC \times 0.75 + IC}{IC} = 1.75$ faster

3. Ideal CPI = 2, clock cycle Time = 1ns, MPI = 1.5 (memory access per instruction)
cache size = 64K, block size = 64 bytes.
① direct mapped miss rate = 1.4%. ② two-way set associative miss rate = 1.0%. (cycle time = 1.25 × 原 cycle time)
miss penalty: 75ns, hit time: 1 cycle

Q: 两种方式的AMAT和CPU time.

A. AMAT: ① $1 + 0.014 \times 75 = 2.05ns$ ② $1 \times 1.25 + 0.01 \times 75 = 2ns$

CPUtime: ① $IC \times (2 \times 1.0 + (1.5 \times 0.014 \times 75)) = 3.58 \times IC$

② $IC \times (2 \times 1.0 \times 1.25 + (1.5 \times 0.01 \times 75)) = 3.63 \times IC$

用的公式: $IC \times [(CPI_{execute} \times Clock\ Cycle\ time) + (Miss\ rate \times \frac{Memory\ accesses}{Instruction} \times Miss\ penalty \times clock\ cycle\ time)]$

基于AMAT的优化方向:

1.减少miss penalty: multilevel caches. critical word first. read miss before write miss. merging write buffers. victim caches.

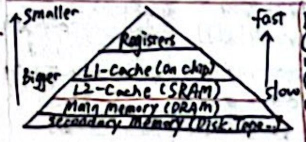
2.减少miss rate: larger block size. larger cache size. higher associativity. way prediction. pseudo-associativity. compiler optimizations

3.减少hit time: small and simple caches. avoiding address translation. pipelined cache access. trace caches.

4.通过并行减少miss rate和miss penalty: non-blocking caches. hardware prefetching. compiler prefetching.

Stack replacement algorithm

	2	3	2	1	5	2	4	5	3	2
St(1)	2	3	2	1	5	2	4	5	3	2
St(2)		2	3	2	1	5	2	4	5	3
St(3)				3	2	1	5	2	4	5
St(4)					3	3	1	1	2	4
St(5)							3	3	1	1
St(6)										



缓存内存设计问题:

①block placement - 为了降低miss rate 采取全相联

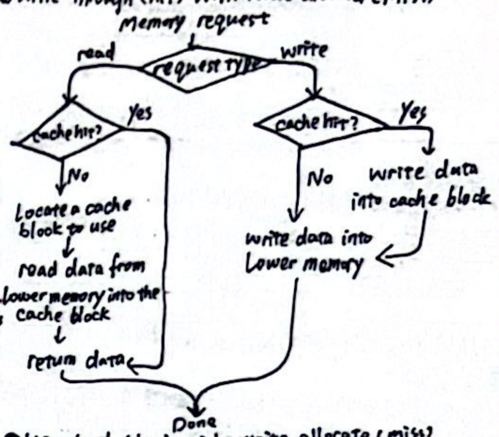
②block indentification - 标号

③block replacement - LRU

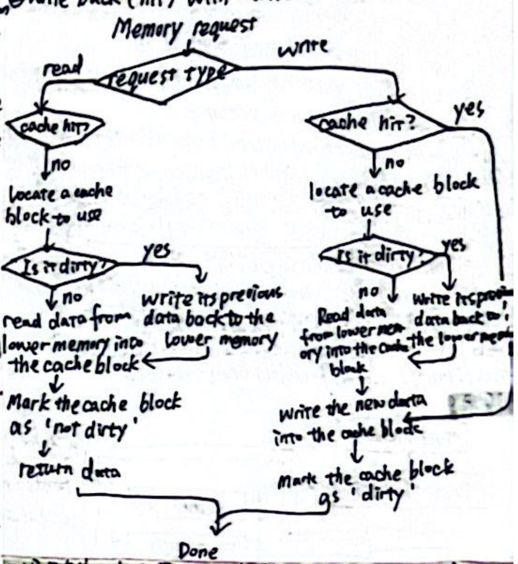
④write strategy - write back with dirty bit. 在不得不写回disk的时候写回.

Write strategy

① Write through (hit) with write around (miss)

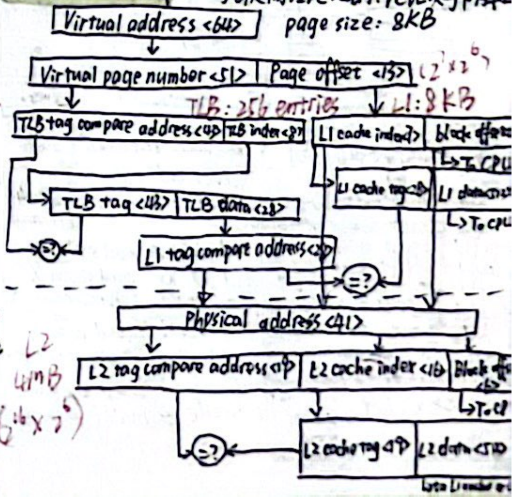


② Write back (hit) with write allocate (miss)



TLB 及多级cache计算

tag都是物理地址的. 地址以字节为单位



多流水线的流水线技术

① Superscalar:

静态调度超标量，通过编译器完成，每个时钟周期发射出一个整型指令和一个浮点数指令，并行执行；如果遇到分支跳转等指令，那么只流出这一条，不能同时发射其他指令一起流出。

动态调度超标量，通过硬件完成，指令顺序进入两个保留站，并行分析处理；对于没有分支预测的情况，需要等待分支预测的结果出来后相关指令再进入Ex阶段。

② Super pipeline 依赖流水线的细分，需要硬件的支持。

③ VLIW：把多条更长的单周期指令并行，通过编译器完成

向量链接计算

基本原则：①没有冲突的指令并行，存在RAW的指令链接，存在结构冲突的指令串行；向量功能内部流水线执行。② 余下部分不满一个向量单独处理

例题：向量加法、读写需要6拍，向量乘法需要7拍，把数据从寄存器送到功能部件需要1拍，功能部件结果写向寄存器需要一拍。

Q：计算 $D=A\times(B+C)$，vector of length N，且B和C已经被存在V0和V1

A: $V3 \leftarrow$ memory // access vector A

$V2 \leftarrow V0+V1$ // vector B and vector C perform floating point addition

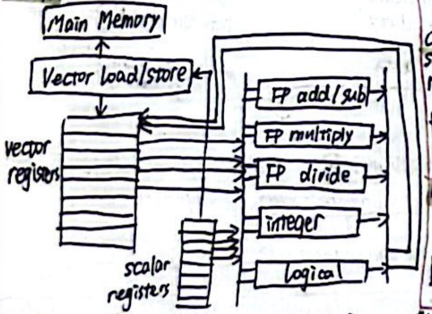
$V4 \leftarrow V2 * V3$ // floating point multiplication, the result is stored in V4

① 三条指令串行的情况：经过对拍V0的第一个元素到达V2，那么再过N-1拍V0的最后一个元素就会到达V2，共：$[(1+6+1)+N-1]+[(1+6+1)+N-1]+[(1+7+1)+N-1]=3N+22$ 拍

② 前两条指令并行和第三条指令串行的情况，共：$\max\{[(1+6+1)+N-1],[(1+6+1)+N-1]\}+[(1+7+1)+N-1]=2N+15$ 拍。

③ 采用向量链接技术：我们只需要知道V4的第一个结果多久可以出来：$8+1+7+1=14$ 拍。随后还有N-1条指令。因此总共需要的拍数为：$\max\{(1+6+1),(1+6+1)\}+(1+7+1)+N-1=N+16$ 拍。

RV64V



标量处理和向量处理的区别：DAXPY (Double Precision a*X plus Y)

RISC-V (scalar)

```
      fld   f0,a          #load scalar a
      addi  x28,x5,#256   #last address to load
Loop: fld   f1,0(x5)      #load X[i]
      fmul.d f1,f1,f0     #a x X[i]
      fld   f2,0(x6)      #load Y[i]
      fadd.d f2,f2,f1     #a x X[i] + Y[i]
      fsd   f2,0(x6)      #Store into Y[i]
      addi  x5,x5,#8      #increment index to x
      addi  x6,x6,#8      #increment index to y
      bne   x28,x5,Loop   #check if done
```

RV64V code:

```
vsetdcfg 4*FP64    #Enable 4 DPFP vregs
fld     f0,a       #load scalar a
vld     v0,x5      #load vector X
vmul    v1,v0,f0   #vector-scalar mult
vld     v2,x6      #load vector Y
vadd    v3,v1,v2   #vector-vector add.
vst     v3,x6      #Store the sum
vdisable           #disable vector regs
```

structural hazards

LLP

① 没有跨迭代相关的情况，可并行执行：for(i=999; i>=0; i=i-1) x[i]=x[i]+s;

② 有跨迭代相关但是可以消除：

```
for (i=0; i<100; i=i+1) {
    A[i] = A[i] + B[i];     /* S1 */
    B[i+1] = C[i] + D[i];   /* S2 */ }
```

交换S1, S2，随后把第一次和最后一次运算提出去，改为：

```
A[0] = A[0]+B[0];
for (i=0; i<99; i=i+1) {
    B[i+1] = C[i] + D[i];     /* S2 */
    A[i+1] = A[i+1]+B[i+1];   /* S1 */ }
B[100] = C[99] + D[99];
```

消除跨迭代相关的依据是不让B[i]和B[i+1]出现在一个循环里，把两次迭代换到同一个循环内；本次循环的两个指令相关可以使用向量链接技术解决。

③ 无法消除的跨迭代循环：(两条指令都是跨迭代相关)

```
for (i=0; i<100; i=i+1) { A[i+1]=A[i]+C[i];   /* S1 */
                          B[i+1]=B[i]+A[i+1]; /* S2 */ }
```

renaming

```
for(i=0; i<100; i=i+1){           for(i=0; i<100; i=i+1){
  Y[i] = X[i]/C;                     T[i] = X[i]/C;
  X[i] = X[i]+C;           =>        P[i] = X[i]+C;
  Z[i] = Y[i]+C;                     Z[i] = T[i]+C;
  Y[i] = C-Y[i]; }                   Y[i] = c - T[i]; }
```

向量计算相关术语

护航指令组convoy：不包含任何结构冒险的可以在同一周期开始执行的向量指令集合；链接chain：允许向量操作在其向量源操作数的各个元素可用时立即启动，链接中第一个功能单元的结果被转发给第二个功能单元；钟鸣Chime：执行护航指令组所花费的时间单位；执行m个护航指令需要m次钟鸣，向量长度为n时需花费m×n个时钟周期。多车道Multi-Lane：多个功能单元来提高性能，单车道1条向量指令需要64周期完成，若变成4车道则只需要16周期完成。

Vector execution time 例子：

how many convoys/chimes? 3

convoy1: vld, vmul (v0 dependence)

convoy2: vld, vadd (v2 dependence)

convoy3: vst

cycles per FLOP = $\frac{3}{2}=1.5$

convoy 内-链接，一个convoy内指令可以并行，3个chime 3个顺序

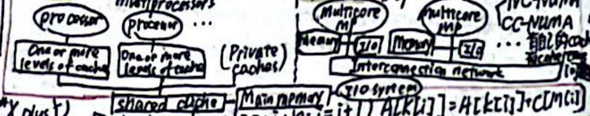
consistency & coherence

cache coherence defines the behavior of reads and writes to the same memory location; Memory consistency defines the behavior of reads and writes with respect to accesses to other memory locations.

Sequential consistency：严格限制读写顺序

Relaxed Consistency Model:
- Total store ordering: Relax W→R
- Partial store order: Relax W→W
- Weak ordering and release consistency: Relax R→W and R→R

UMA / SMP / centralized shared-memory multiprocessors | NUMA / DSP



另个例子：for(i=0; i<n; i=i+1) A[K[i]] = A[K[i]] + C[M[i]]

use index vector:

```
vsetdcfg 4*FP64      # 4 64b FP vector registers
vld     v0,x7        # load K[]
vldx    v1,x5,v0     # load A[K[]]
vld     v2,x28       # load M[]
vldi    v3,x6,v2     # load C[M[]]
vadd    v1,v1,v3     # Add them
vstx    v1,x5,v0     # store A[K[]]
vdisable             # Disable vector registers
```

和向量机相同之处：①可以同时处理数据级并行问题 ② Scatter-gather transfers ③ large register files ④ Mask registers 和向量机不同之处：①没有标量处理机 ②采用多线程隐藏内存延迟 ③拥有多个功能单元，而不像向量处理机那样流水线 ④增加了硬件实现的分支调度机制，更长

NVIDIA GPU

GPU → Grid → Thread block → SIMD CUDA thread 逐个执行

GPU memory　per-block local memory　per-CUDA thread private memory

Snoopy Protocols 侦听

① Write Invalidate Protocol：多reader，单writer，写入共享数据的时候，将一个(Invalidate)信号送到所有的cache，这个信号将所有对应地址的cache line失效。因此下一下读取的时候会出现read miss，出现read miss时，如果使用write through，直接从mem中读取即可；如果使用write-back，则在所有cache中寻找最新的数据。

② Write Broadcast Protocol：多为write-through情况，当一个处理器写入数据时，使用bus广播，将所有Cache中对应的数据更新

③ Write serialization Protocol：使用总线进行序列化。

tomasulo with speculation

乱序执行，顺序提交，在Tomasulo的基础上添加ReOrderBuffer，取消store buffer，通过ROB换名，RS仅在issue和exe之间保存op和操作数，ROB负责exe结束到commit。

Commit：非跳转(更新寄存器或mem，移除ROB项)；错误跳转(刷新ROB，从正确PC重新开始)

scoreboard 停顿于 issue(结构竞争和WAW) WB(WAR);

tomasulo停顿于 issue (RS结构竞争)

带speculation的tomasulo停顿于 issue (RS和ROB结构竞争)

Cache计算

VA=46 bits. PA=36 bits. Page size=8KB, Cache Capacity =16KB. an 8-way physically addressed cache.

Q: tag in the 16 KB cache is ___ bits?

A: 对于PA，tag+index+offset=36 bits.

设一个cache line大小为 2^x 字节，共 2^{14-x} 个cacheline ⇒ 2^{11-x} 项，index为 11-x bits，offset为 x bits ⇒ tag = 36-(11-x+x)=25

内存计算

要计算：$d_i=a_i+b_i*c_i$ (i: 0~511)，一个元素4-byte-wide

Mem. address in bytes	Contents
0~2047	Array a
2048~4095	Array b
4096~6143	Array c
6144~8191	Array d

write back & allocate cache: 32k byte, 4-way associative. hit time: 1 cycle, miss penalty & write back time: 100 cycles.

Qa. 一个iter几个cycles 如果ld和sd全miss?

Aa. 假设指令执行时间(不包括访存)为t(一个iter)：$400+4+t$

Qb. cache line大小为16 bytes，一个iter执行的平均cycle?

Ab. 一次miss可使4个元素hit：$\frac{400}{4}+4+t$，由于4-way故无replacement

Qc. cacheline大小为64 bytes，一个iter执行的平均cycle?

Ac. $\frac{400}{16}+4+t$

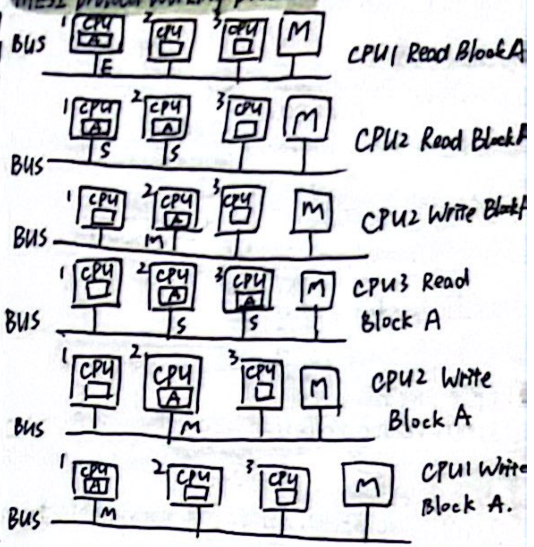
Qd. 如果cache：direct-mapped且大小变为2048 bytes，问一个iter执行的平均cycle?

Ad. 假设cache line 2^k bytes，能放下 2^{k-2} 个数组元素，即 $a_0, a_1, \dots, a_{2^{k-2}-1}$ 会在同一个cache line里，且 ai bi ci di 映射到同一个cache line，要在miss penalty的基础上加上di write back的代价，但 2^{k-2} 迭代过去后，此时ai bi ci di从当前cache line移动到下一个cache line，因此当前cache line最后存入的di就不会被write back：$t+4+400+(1-\frac{1}{2^{k-2}})\times 100$

	Busy	Instruction
Load1	no	
Load2	no	
load3		

	F0	F2	F4	F6	F8	F10 …	F30
ROB	#3			#6	#4	#5	
Busy	yes	no	no	yes	Yes	Yes …	no

MESI protocol working process



Scoreboard 与 Tomasulo

Function Component Status

Fi-目的操作数对应的寄存器；Fj、Fk-源操作数对应的寄存器

Qj、Qk-源操作数等待的部件

Rj、Rk-源操作数的状态，仅理解这应该是一个寄存器的"锁"。

Yes-这个操作数已经准备好了但没有读，因为其他操作数还没有ready，此时其他指令不能WB这个寄存器。

no-其他指令可以WB这个寄存器

Qi/Qj == NULL-操作数已读取完毕

Qi/Qj != NULL-正在等待Q中所指的部件的执行，操作数未准备

	IS	RO	EX	WB
FLD F6,34(R2)	1	2	3	4
FLD F2,45(R3)	5	6	7	8
FMUL.D F0,F2,F4	6	9	10~19	20
FSUB.D F8,F2,F6	7	9	10~11	12
FDIV.D F10,F0,F6	8	21	22~61	62
FADD.D F6,F8,F2	13	14	15~16	22

Tomasulo Reservation Stations Table:

Vj、Vk-记录操作数具体的值，Qj、Qk-记录操作数等待的保留站

A-记录ld、sd指令的取值/写入地址。

Tomasulo with speculation

Instruction	IS	Ex	WB	Commit
FLD F6,34(R2)	1	3	4	5
FLD F2,45(R3)	2	4	5	6
FMUL.D F0,F2,F4	3	6~15	16	17
FSUB.D F8,F6,F2	4	6~7	8	18
FDIV.D F10,F0,F6	5	17~56	57	58
FADD.D F6,F8,F2	6	9~10	11	59

Reservation stations:

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	yes	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add		Regs[F2]	#4		#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	DIV		Regs[F6]	#3		#5	

Reorder buffer:

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6,34(R2)	commit	F6	Mem[load1]
2	no	FLD F2,45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0,F2,F4	Ex1	F0	
4	Yes	FSUB.D F8,F6,F2	Ex1	F8	
5	Yes	FDIV.D F10,F0,F6	Issue	F10	
6	Yes	FADD.D F6,F8,F2	Issue	F6	