University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1997 D.A. Patterson
Solution #3b

Assignment 2
Solution Set
CS152 Computer Architecture and Engineering

3.14 Instruction Reads = $1/instruction$
Data Reads = $\frac{2}{3} * .33/instruction$
Data Reads = $\frac{1}{3} * .33/instruction$

Total Memory Accesses = $InstructionReads + DataReads + DataWrites = 1.33/instruction$
Total Instruction Memory Accesses = $InstructionReads = 1/instruction$
Total Data Memory Accesses = $DataReads + DataWrites = .33/instruction$
Total Read Memory Accesses = $InstructionReads + DataReads = 1.22/instruction$
Total Write Memory Accesses = $DataWrites = .11/instruction$

a. $\frac{.33}{1.33} = .24$ or 24% of accesses for data
b. $\frac{1.22}{1.33} = .91$ or 91% of accesses are reads

3.19 Initially all data in memory. At the end all results in memory.

- **Accumulator**:

	Instruction	Instr bytes	Data Bytes
1.	load addressB	1+2=3	4
2.	add addressC	3	4
3.	store addressA	3	4
4.	add addressC	3	4
5.	store addressB	3	4
6.	load addressA	3	4
7.	sub addressB	3	4
8.	store addressD	3	4
	Total	24	32

Each instruction is 3 bytes long (1 byte for opcode, 2 for address), so 24 bytes of code fetched.
Each instruction loads or stores 4 bytes of data, so 32 bytes of data transfered.

- **Memory-memory**:

	Instruction	Instr bytes	Data Bytes
1.	add addressA, addressB, addressC	1+2+2+2=7	3x4=12
2.	add addressB, addressA, addressC	7	12
3.	sub addressD, addressA, addressB	7	12
	Total	21	36

Each instruction is 7 bytes long (1 opcode, 3 address), so 21 bytes of code fetched. Each instruction loads 8 bytes and stores 4, so 36 bytes of data transfered.

- **Stack**:

	Instruction	Instr bytes	Data Bytes
1.	push addressB	1+2	4
2.	push addressC	3	4
3.	add	1	0
4.	pop addressA	3	4
5.	push addressA	3	4
6.	push addressC	3	4
7.	add	1	0
8.	pop addressB	3	4
9.	push addressA	3	4
10.	push addressB	3	4
11.	add	1	0
12.	pop addressA D	3	4
	Total	30	36

Push and pop instructions are 3 bytes long, add and sub just 1 byte. So, 30 bytes of code fetched. Each push and pop transfers 4 bytes of data, so 36 bytes of data transfered.

- **Load-store**:

	Instruction	Instr bytes	Data Bytes
1.	load $r1, addressB	1+.5+2=4	4
2.	load $r2, addressC	4	4
3.	add $r3, $r2, $r1	1+.5+.5+.5=3	0
4.	add $r2, $r3, $r2	3	0
5.	sub $r4, $r3, $r1	3	0
6.	store $r1, addressB	4	4
7.	store $r3, addressA	4	4
8.	store $r4, addressD	4	4
	Total	29	20

Each load/store instruction is 4 bytes long, each add/sub is 3 bytes long. So, 29 bytes of code fetched. Each load or store transfers 4 bytes of data, so 20 bytes of data transfered.

The memory-memory architecture is the most efficient in terms of code size. The load-store architecture is the most efficient in terms of total bytes fetched. Memory-memory has very powerful instructions but has to fetch all operands from memory. Load-store takes advantage of registers in order to reuse operands fetched as long as they are not overwritten.

A.3 These registers are considered by the OS as unused by user programs so it never saves and restores them. If using any OS and there is a chance that the user program will get interrupted

(I/O interrupt, fault, context switch etc) before full completion they should not be used. If the system is such that a user program is guaranteed to complete before without any interruption, it is ok to use them.