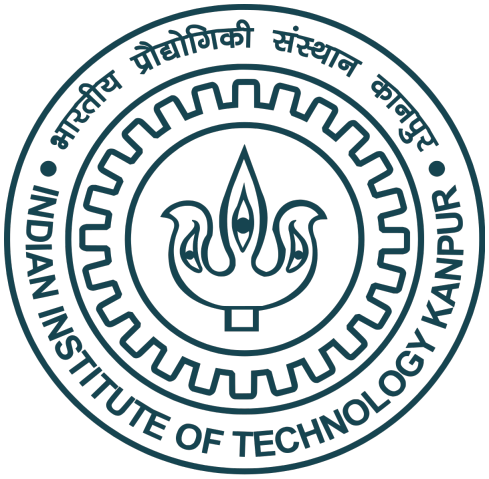
CS422: COMPUTER ARCHITECTURE



Assignment 3

Parinay Chauhan
200667

November 10, 2023

1 PART A:

Test Case	Instructions	Loads (%)	Stores (%)	Conditional Branches (%)
asm-sim	158	39.24050633	34.81012658	0
c-sim	1928	17.42738589	8.195020747	19.29460581
endian	1965	19.13486005	11.043257	18.01526718
factorial	2530	16.48221344	8.616600791	18.41897233
fib	44903	13.5915195	17.77609514	8.344654032
hello	1744	19.0940367	8.600917431	19.61009174
host	14894	20.10205452	9.923459111	19.22250571
ifactorial	2420	16.48760331	7.851239669	19.25619835
ifib	7581	16.98984303	10.01187178	17.50428703
log2	2096	17.89122137	8.349236641	19.60877863
msort	17219	10.58714211	5.441663279	18.12532667
rfib	25668	14.39925199	17.02898551	9.513791491
subreg	3892	17.75436793	8.170606372	19.50154162
towers	82763	20.05727197	8.488092505	20.54299627
vadd	7259	6.144096983	16.35211462	10.57996969

2 PART B:

2.1 Results:

The designs mentioned below represent the following designs:

- **Design-1:** The complete interlock logic design with no bypass paths and two delay slots for branches.
- **Design-2:** Design-1 but with only one delay slot for branches.
- **Bypass-1:** Design-2 but with EX-EX bypass path enabled
- **Bypass-2:** Bypass-1 with MEM-MEM bypass path enabled
- **Final Design:** This is the final design with only one delay slot for branches and all possible bypass paths (EX-EX, MEM-MEM and MEM-EX) enabled.

The **CPI** of all these designs are indicated below.

Test Case	Design-1	Design-2	Bypass-1	Bypass-2	Final Design	Load-delay stalls(%)
asm-sim	1.57	1.47	1.22	1.22	1.22	0
c-sim	1.8	1.65	1.11	1.11	1.01	0
endian	1.73	1.58	1.11	1.1	1.01	0
factorial	1.8	1.63	1.1	1.1	1.01	0
fib	1.63	1.41	1.07	1.07	1	0
hello	1.77	1.62	1.12	1.12	1.01	0
host	1.62	1.45	1.11	1.1	1.01	0
ifactorial	1.8	1.65	1.1	1.1	1.01	0
ifib	1.68	1.54	1.09	1.08	1	0
log2	1.75	1.59	1.11	1.11	1.01	0
msort	1.86	1.72	1.04	1.04	1	0
rfib	1.63	1.43	1.07	1.07	1	0
subreg	1.76	1.59	1.1	1.1	1.01	0
towers	1.59	1.41	1.1	1.09	1	0
vadd	1.97	1.88	1.04	1.04	1	0

2.2 Approach and Analysis

To implement the final design the following steps were followed:

- To remove one branch delay slot, the branch target calculation was implemented in a phased manner with the execute checking if the current instruction is a branch instruction (by checking the value of the $_bd$ variable. If true the target was calculated in the first half of the execute cycle and the new pc was read by the fetcher in the second half.
- To implement the bypass paths, a buffer states of all logical registers were created. Value was written to these buffer states as soon as it was calculated. Moreover, the instructions with such registers as their source registers always read from the buffer states when executing the instructions.
- If certain instruction had to be stalled due to unavailability of the source register value, it was stalled in the decode phase itself. This was implemented by storing the number of cycles it would take for the value of the destination register, of every instruction that entered the decode phase to become available. This counter was decreased every cycle. Thus for every incoming instruction, the availability of it's source registers could be checked in the decode phase itself.

.
Some highlights of the design and stats:

- The phased writing and reading of registers was implemented at the start itself thus all designs presented above include phased register read and write.
- When a syscall is encountered, the whole pipeline is stalled till it reaches the write-back phase and is executed. The instruction that has already been fetched is also discarded. This leads to 4 stall cycles for each syscall.
- The stall cycles of syscalls are the reason for the some test cases having CPI > 1. Below is a table representing the percentage of syscalls for each test case. The table is indicative of the reason behind the CPI being greater than 1.

Test Case	**Number of Instructions**	**Number of Syscalls**	**Percentage Syscalls (%)**
asm-sim	49	3	6.12244898
c-sim	1825	5	0.2739726027
endian	1864	5	0.2682403433
factorial	2407	5	0.2077274616
fib	44779	5	0.01116594832
hello	1638	5	0.3052503053
host	14519	24	0.1653006405
ifactorial	2293	5	0.2180549498
ifib	7453	5	0.06708707903
log2	1995	5	0.2506265664
msort	17103	5	0.0292346372
rfib	25540	5	0.01957713391
subreg	3738	7	0.1872659176
towers	80651	67	0.08307398544
vadd	7129	5	0.07013606396