Assignment 1

Analyze `ia-32` compiled SPEC-2006 Benchmark Suite

Anshuman Barnwal · 200259

Homework Assignment of

CS422

Monday 25th September, 2023

BACKGROUND

You will analyze a subset of the SPEC 2006 benchmark suite compiled for the `ia32` (32-bit Intel architecture) ISA. Specifically, you will instrument each SPEC 2006 application binary with PIN and extract the percentages of various different instruction types. You will also use a very simple cycle accounting model to calculate the CPI for each application. Additionally, you will calculate the instruction and data footprints of each application. Finally, you will understand a few properties of the `ia32` ISA.

In this assignment, we will be interested only in the following instruction types. At a very high level, we have two instruction types.

Type A: Instructions with no memory operand
Type B: Instructions with at least one memory operand

You should only instrument the instructions that actually execute i.e., have true predicates. As a result, you should always use `INS_InsertPredicatedCall` method when classifying instructions.

For Type B Instructions, Each memory operand in such an instruction can be read and/or written to.

We will view each such load or store operation within a type B instruction as a separate micro-instruction and count it as a separate instruction.

For example, a type B instruction may have two load operations and a store operation. Further, each such load or store operation may access an arbitrary amount of data. However, in a 32-bit processor, it is not possible to transfer more than 32 bits of data in one shot. As a result, we will further break down each memory operation into several memory accesses of size at most 32 bits.

Finally, the instruction must be using these memory operands to carry out some operation of type A. This operation should be counted as a separate type A instruction and should also be categorized into one of the fifteen categories mentioned above.

Let us take an example of an x86 instruction that has three memory operands, two of which are loads and one is a store. The loads access one and ten bytes, respectively. The store accesses eleven bytes. This instruction will be accounted as follows.

```
Number of loads: 4
Number of stores: 3
Number of type A instruction: 1
```

Total instruction count increases by eight, whenever such an instruction is encountered.

TASKS

Part 1

For each benchmark application, you need to report the dynamic counts and percentages of the following seventeen types of instructions. The total number of instructions (to be used as the denominator in the percentage) is the addition of all these seventeen counters.

Prepare a table showing these counts and percentages for the applications.

Solution.

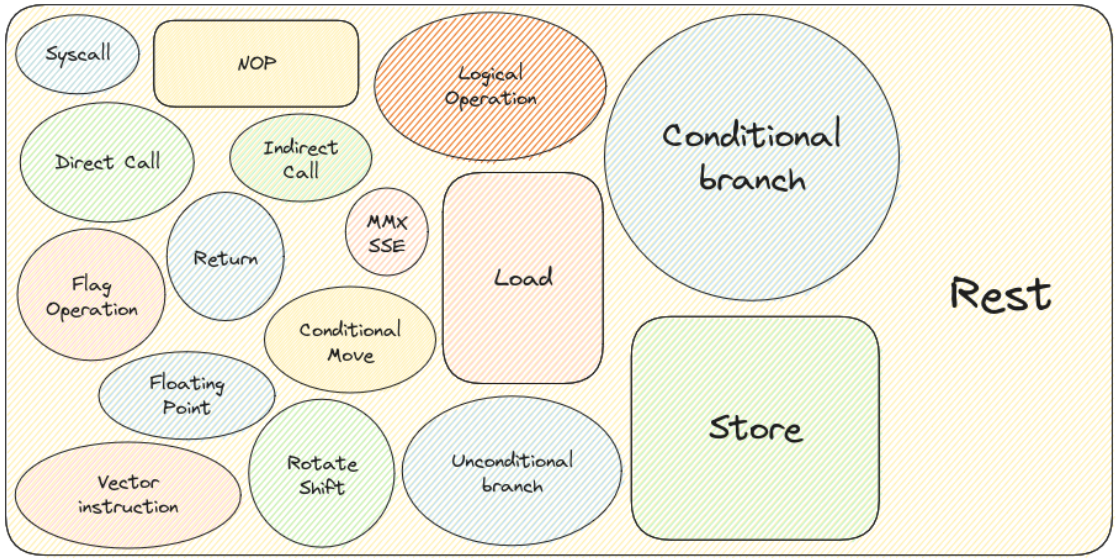

Figure 1: Types of Instructions (Load-Store may overlap others)

All the benchmark applications provided were fast-forwarded by a given number of instructions. During fast-forwarding, consider each instruction irrespective of type, as 1 instruction while in instrumentation period, considered every instruction containing n 32b loads and m 32b stores components as equivalent to $1 + n + m$ instructions.

The below tables store the required data regarding all type of instructions. "Percent" is the corresponding $\frac{\text{Count}}{\text{Effective}}$ where "Effective" is as in 1.

We found "Vector Instructions", "Conditional Moves", "MMX and SSE Instructions", and "System Calls" to be 0 for all the applications.

Below at 6, Memory Instructions is essentially Loads and Stores together.

Table 1: Actual #Instruction v/s Effective $1 + n + m$ #Instruction and their ratio

Application	FastForward$^{(B)}$	Instruction Counts		Percentage
		Actual	Effective	
400.perlbench	207	643629926	999864910	64.37%
401.bzip2	301	626757711	999997068	62.67%
403.gcc	107	694478005	999992310	69.45%
429.mcf	377	660698717	1000000000	66.07%
450.soplex	364	615283889	1000000001	61.53%
456.hmmer	264	616007786	999999444	61.60%
471.omnetpp	43	625784586	1000000000	62.58%
483.xalancbmk	1331	652390226	999688603	65.26%

Table 2: Load and Store #Instructions

Application	Loads		Stores	
	Count	Percent	Count	Percent
400.perlbench	235396718	23.543	120973356	12.099
401.bzip2	286663856	28.666	86578433	8.658
403.gcc	22555350	2.255	282966645	28.301
429.mcf	274762679	27.476	64538604	6.459
450.soplex	335145819	33.514	49570293	4.957
456.hmmer	337431712	33.743	46560502	4.656
471.omnetpp	232166088	23.217	142049326	14.205
483.xalancbmk	239302390	23.938	108307384	10.834

Table 3: NOP, Direct and Indirect Calls #Instructions

Application	NOPs		Direct Calls		Indirect Calls	
	Count	Percent	Count	Percent	Count	Percent
400.perlbench	433528	0.0433	6947662	0.695	2165028	0.216
401.bzip2	32147	0.00321	11032	0.00110	0	0.000
403.gcc	90112	0.00901	1275299	0.128	13555	0.00136
429.mcf	879752	0.0879	8266384	0.826	0	0.000
450.soplex	1852	0.00018	2374689	0.237	54	0.0000
456.hmmer	14312	0.0014	86688	0.0087	553	0.00005
471.omnetpp	503202	0.0503	13391515	1.339	2307982	0.231
483.xalancbmk	14237079	1.424	8707590	0.871	5826909	0.583

Table 4: Returns, Unconditional and Conditional Branches #Instructions

Application	Returns		Uncond. Branches		Cond. Branches	
	Count	Percent	Count	Percent	Count	Percent
400.perlbench	9112689	0.911	20172634	2.017	82678603	8.268
401.bzip2	11031	0.0011	14741341	1.474	84526720	8.453
403.gcc	1288855	0.128	1488031	0.148	97945541	9.794
429.mcf	8266383	0.826	5572158	0.557	117749486	11.77
450.soplex	2374743	0.237	7571500	0.757	64241385	6.424
456.hmmer	87241	0.0087	113915	0.0113	88919715	8.891
471.omnetpp	15699500	1.569	13881657	1.388	73391837	7.339
483.xalancbmk	14538758	1.454	5687717	0.568	115456999	11.549

Table 5: Logical, Rotate-Shift and Flag Operations #Instructions

Application	Logical Op		Rotate & Shift		Flag Op	
	Count	Percent	Count	Percent	Count	Percent
400.perlbench	61151842	6.116	2897866	0.289	478094	0.0478
401.bzip2	51058392	5.105	44677195	4.467	4398	0.0004
403.gcc	94626225	9.462	917464	0.0917	36715	0.0036
429.mcf	49278688	4.927	2313100	0.231	0	0.000
450.soplex	8780117	0.878	6691975	0.669	13408915	1.340
456.hmmer	692045	0.0692	167329	0.0167	2879	0.0002
471.omnetpp	37629094	3.762	4483798	0.448	12598264	1.259
483.xalancbmk	24767577	2.477	3702529	0.370	1093111	0.109

Table 6: Floating Point, Rest of Instructions, and Total Memory Operations #Instructions

Application	Floating Point		The Rest		Memory Instructions	
	Count	Percent	Count	Percent	Count	Percent
400.perlbench	342346	0.0342	457114544	45.717	351643026	35.169
401.bzip2	0	0.000	431692523	43.169	371409789	37.141
403.gcc	0	0.000	496788518	49.679	305508440	30.551
429.mcf	0	0.000	468372766	46.837	339301283	33.930
450.soplex	186666057	18.666	323172602	32.317	291902950	29.190
456.hmmer	17169	0.00171	525905384	52.590	383971218	38.397
471.omnetpp	60608252	6.060	391289485	39.128	344614207	34.461
483.xalancbmk	4770381	0.477	453290179	45.343	330430323	33.053

Part 2

The second part of the assignment involves calculating the CPI for each benchmark. You should charge each load and store operation a fixed latency of seventy cycles and every other instruction a latency of one cycle.

Tabulate the CPI of the applications in a table.

Solution.

This can be simply calculated by

$$\begin{aligned} \texttt{CPI} &= \texttt{Memory Type\% * 70 + Rest Type\% * 1} \\ &= \texttt{Memory Type\% * 69 + 1} \end{aligned} \tag{1}$$

Table 7: CPI per application

Application	`Memory Type%`	CPI
400.perlbench	35.169	24.266
401.bzip2	37.141	25.627
403.gcc	30.551	21.080
429.mcf	33.930	23.412
450.soplex	29.190	20.141
456.hmmer	38.397	26.494
471.omnetpp	34.461	23.778
483.xalancbmk	33.053	22.806

Part 3

For each application benchmark, you need to calculate the number of unique 32B instruction and data chunks accessed. Report the instruction and data statistics separately.

In this assignment, we will calculate the memory footprint at a granularity of 32B. Prepare a table showing the instruction and data footprints of the applications.

Solution.

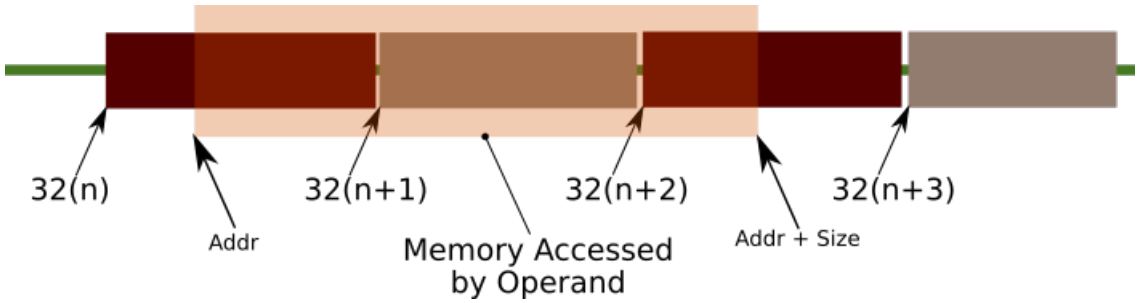

Figure 2: Schematic representation of Memory access depicting granularity

In the figure 2 all of $32n$, $32(n+1)$ and $32(n+2)$ must be considered for that particular memory access.

Using the above interpretation, we consider memory chunks accessed by the program with respect to both instruction and data memory. Following are the results:

Table 8: Instruction and Data Memory Chunks accessed

Application	I. Memory Chunks	D. Memory Chunks
400.perlbench	2849	21106
401.bzip2	272	317909
403.gcc	996	438547
429.mcf	64	7732611
450.soplex	625	5661102
456.hmmer	462	84599
471.omnetpp	901	529903
483.xalancbmk	2285	595797

Part 4

Tabulate the results for the applications appropriately:

1. Distribution of instruction length
2. Distribution of the number of operands in an instruction
3. Distribution of the number of register read operands in an instruction
4. Distribution of the number of register write operands in an instruction
5. Distribution of the number of memory operands in an instruction
6. Distribution of the number of memory read operands in an instruction
7. Distribution of the number of memory write operands in an instruction
8. Report the maximum and average number of R-W memory bytes touched
9. Report the maximum and minimum values of the immediate field in an instruction
10. Report the maximum and minimum values of the displacement field in a memory

Solution.

Due to large count size and overflow in showing them in a single table, we chose to just display the percentage for showing distributions of above. Therefore for some 0.000 might not mean that quantity is exactly 0. To see the exact value, refer to output files.

1. Table 9 represents the distribution of instruction length. Note that columns from $1 \cdots 10$ represent number of bytes, while `App` column represents the same benchmark applications as before. All counts are in percentage of all instructions.
2. Table 10 represents the distribution of number of operands. The above format still applies here.
3. Table 11 represents the distribution of number of register read operands.
4. Table 12 represents the distribution of number of register write operands.
5. ;
6. ;
7. Table 13 represents the distribution of number of memory operands, containing 3 major columns "#Memory Operands" with possible 1 or 2 operands, "#Memory Read Operands" and "#Memory Write Operands" with possible 0, 1 or 2 operands.
8. ;
9. ;
10. Table 14 contains statistics regarding Number of Bytes accessed (both read and write), Immediate Field and Displacement Field, all per Operand.

Table 9: Instruction Size in Percent

App	1	2	3	4	5	6	7	8	9	10
400	11.356	25.172	27.123	5.182	7.879	19.868	3.419	0.000	0.000	0.000
401	3.923	20.049	47.280	7.104	1.817	11.990	5.995	1.842	0.000	0.000
403	14.297	67.261	3.119	13.625	0.566	0.832	0.293	0.006	0.000	0.000
429	8.057	48.355	31.647	5.040	2.246	0.561	4.095	0.000	0.000	0.000
450	7.917	43.613	40.094	1.947	0.390	3.805	1.851	0.384	0.000	0.000
456	2.501	30.291	29.576	27.071	2.479	6.852	0.041	1.189	0.000	0.000
471	15.486	30.902	38.157	3.436	4.510	4.859	2.650	0.000	0.000	0.000
483	14.559	31.906	44.701	3.211	2.423	2.384	0.776	0.027	0.007	0.007

Table 10: Number of Operands in Percent

App	0	1	2	3	4	5	6
400	0.067	0.102	52.849	35.476	9.703	1.416	0.387
401	0.005	0.001	59.839	39.854	0.007	0.002	0.293
403	0.013	0.022	30.419	42.029	1.271	26.246	0.000
429	0.133	0.000	48.525	45.723	4.368	1.251	0.000
450	0.000	0.000	41.793	39.317	18.501	0.388	0.000
456	0.002	0.000	56.621	43.291	0.068	0.015	0.003
471	0.080	0.023	51.812	28.168	17.290	2.510	0.116
483	2.182	0.075	42.491	41.121	10.252	2.425	1.454

Table 11: Number of Register Read Operands in Percent

App	0	1	2	3	4	5	6
400	0.760	26.897	52.845	17.899	0.888	0.324	0.387
401	0.549	18.542	54.361	21.864	4.391	0.000	0.293
403	0.096	15.416	43.685	1.802	12.938	26.063	0.000
429	0.375	14.871	67.642	17.112	0.000	0.000	0.000
450	2.195	21.459	57.549	14.342	4.451	0.003	0.000
456	0.055	7.556	56.245	28.127	8.014	0.001	0.003
471	2.900	19.771	54.017	21.983	0.845	0.368	0.116
483	2.922	23.632	45.950	25.037	0.124	0.881	1.454

Table 12: Number of Register Write Operands in Percent

App	0	1	2	3	4	5	6
400	13.109	69.059	17.446	0.261	0.126	0.000	0.000
401	13.522	72.248	13.937	0.293	0.000	0.000	0.000
403	6.783	72.419	20.797	0.000	0.000	0.000	0.000
429	7.094	77.037	15.866	0.004	0.000	0.000	0.000
450	7.375	75.942	16.683	0.000	0.000	0.000	0.000
456	7.518	75.531	16.948	0.003	0.000	0.000	0.000
471	16.531	66.530	16.785	0.038	0.116	0.000	0.000
483	11.406	68.527	18.614	1.454	0.000	0.000	0.000

Table 13: Number of Memory, Memory Read and Memory Write Operands in Percent

App	#Memory Operands		#Memory Read Operands			#Memory Write Operands		
	1	2	0	1	2	0	1	2
400	98.723	1.277	33.307	66.463	0.230	65.646	34.354	0.000
401	99.507	0.493	22.817	77.183	0.000	76.689	23.311	0.000
403	99.996	0.004	92.617	7.383	0.000	7.378	92.622	0.000
429	100.000	0.000	19.021	80.979	0.000	80.979	19.021	0.000
450	100.000	0.000	12.607	87.393	0.000	87.393	12.607	0.000
456	99.996	0.004	12.121	87.879	0.000	87.874	12.126	0.000
471	99.120	0.880	38.604	61.185	0.211	60.727	39.273	0.000
483	95.488	4.512	28.041	71.959	0.000	67.448	32.552	0.000

Table 14: Bytes Accessed per Operand, Immediate Field and Displacement Field per Operand

App	Bytes per Operand		Immediate Field		Displacement Field	
	max	avg	max	min	max	min
400	8	3.718	2147483647	-2147483648	135918104	-1408
401	8	3.507	100000	-5	134997992	-172
403	8	3.971	138466100	-2147483587	138633083	-1744
429	4	4.0	1374389535	-100000000	134957120	-76
450	8	5.272	2147483647	-640172613	135855532	-344
456	8	3.998	2147483647	-987654321	135294312	-580
471	8	4.221	2147483647	-2092037281	136090116	-104
483	8	4.15	2147483647	-1431655765	139655605	-1392

Sidenotes

- `TARGE_IA32` is a preprocessor directive that is `true` when building for `ia32`
- All instructions that invoke the instrumentation routine are always classified at least as one of the Type-A. They can then be categorised in Type-B as well if they contain memory operands, and each 32B memory dealt with is treated as yet another purely Type-B instruction.
- There happens no categorisation of Type-A/B till the fast-forwarded amount of actual instructions are executed.

32b	32 bits
32B	32 Bytes
(B)	in Billions
#Instructions	Number of Instructions

Table 15: Abbreviations

Please find more information regarding the submission in `README.md` in the submission directory.