ASSIGNMENT-1

CS422A

Hitesh Anand
200449

Question 1, 2

Implementation Details

File Convention

The code producing desired outputs for both the questions is in the file task_1_2.cpp (Run this to get the output corresponding to both the parts)

Initialization

Initialize the ins_counts array which stores the count for each type of instruction. Also, initialize the cycle_counts array which stores the number of cycles for each kind of instruction (70 for LOAD/STORE), and 1 for every other instruction.

```
for (int i = 0; i < TOTAL; i++)
    ins_counts[i] = 0;

for (int i = 0; i < TOTAL; i++)
{
    if (i == LOAD || i == STORE)
        cycle_counts[i] = 70;
    else
        cycle_counts[i] = 1;
}
```

Analysis Routines / Other Functions

- InsCount(): increments the instruction count for every instruction
- FastForward(): checks whether the instruction count has reached the fast forward amount of instructions for the given application

- MyPredicatedAnalysis(): increments the given pointer
- Terminate(): calls the Exit routine when instruction count exceeds fast_forward + BILLION instructions
- Finally, the exit routine computes the CPI and prints all the recorded values

```
void InsCount()     //Increment the instruction count
{
    icount++;
}

ADDRINT FastForward(void)   //Check whether we have reached the fast forward number of instructions
{
    return ((icount >= fast_forward) && (icount < fast_forward + BILLION));
}

void MyPredicatedAnalysis(void *cnt)    //Increment the value of the passed pointer
{
    double *cntr = (double *)cnt;
    (*cntr)++;
}

ADDRINT Terminate(void)     //Check whether we have instrumented BILLION instructions successfully
{
    return (icount >= fast_forward + BILLION);
}
```

```
//Computation of CPI
double total_cnt = 0, weighted_cycles = 0;
for (int is = 0; is < TOTAL; is++)
{
    total_cnt += ins_counts[is];
    weighted_cycles += ins_counts[is] * cycle_counts[is];
}
```

Results

400.perlbench

Instruction Type	Number of Instructions	% of Total Instructions
NOP	954493	0.09
Direct Call	1.048×10^7	1.04
Indirect Call	1.189×10^7	0.11
Returns	1.167×10^7	1.167
Unconditional Branches	1.86×10^7	1.864
Conditional Branches	8.51×10^7	8.514
Logical Operations	7.02×10^7	7.021
Rotate and Shift	2.475×10^6	0.2475
Flag Operations	691965	0.069
Vector Instructions	0	0
Conditional Moves	0	0
MMX and SSE Instructions	0	0
System Calls	0	0
Floating Points	1.075×10^6	0.107
Loads	2.178×10^8	21.781
Stores	1.411×10^8	14.12
Other	4.38×10^8	43.84

CPI for this application: **25.772**

401.bzip2

Instruction Type	Number of Instructions	% of Total Instructions
NOP	247	2.47×10^{-5}
Direct Call	3.58×10^{6}	0.36
Indirect Call	0	0
Returns	3.58×10^{6}	0.36
Unconditional Branches	1.08×10^{7}	1.08
Conditional Branches	6.79×10^{7}	6.79
Logical Operations	1.44×10^{7}	1.44
Rotate and Shift	1.17×10^{7}	1.17
Flag Operations	954	9.54×10^{-5}
Vector Instructions	0	0
Conditional Moves	0	0
MMX and SSE Instructions	0	0
System Calls	0	0
Floating Points	0	0
Loads	2.85×10^{8}	28.54
Stores	1.12×10^{8}	11.24
Other	4.9×10^{8}	49

CPI for this application: **28.4483**

403.gcc

Instruction Type	Number of Instructions	% of Total Instructions
NOP	2.62×10^6	0.26
Direct Call	2.03×10^7	2.03
Indirect Call	151773	0.015
Returns	2.04×10^7	2.04
Unconditional Branches	2.11×10^7	2.1
Conditional Branches	9.69×10^7	9.69
Logical Operations	3.4×10^7	3.40
Rotate and Shift	118943	0.011
Flag Operations	0	0
Vector Instructions	0	0
Conditional Moves	0	0
MMX and SSE Instructions	0	0
System Calls	0	0
Floating Points	0	0
Loads	2.24×10^8	22.40
Stores	1.13×10^8	11.32
Other	4.669×10^8	46.69

CPI for this application: **24.2699**

429.mcf

Instruction Type	Number of Instructions	% of Total Instructions
NOP	4	4×10^{-7}
Direct Call	1.69×10^{7}	1.69
Indirect Call	0	0
Returns	1.69×10^{7}	1.69
Unconditional Branches	113176	0.011
Conditional Branches	8.49×10^{7}	8.49
Logical Operations	4.02×10^{7}	4.02
Rotate and Shift	56577	0.005
Flag Operations	0	0
Vector Instructions	0	0
Conditional Moves	0	0
MMX and SSE Instructions	0	0
System Calls	0	0
Floating Points	0	0
Loads	2.823×10^{7}	28.23
Stores	8.48×10^{7}	8.48
Other	4.735×10^{8}	47.35

CPI for this application: **26.3376**

NOTE: The applications 450.soplex, 456.hmmr, 471.omnetpp, and 486.xalancbmk produce an error: *"PIN out of memory: MmapChecked"* for Question 1,2. This could not be debugged and attempts to debug even led to sometimes crashing my system (Ubuntu 22.04 LTS dual boot with Windows 11). Hence, the outputs corresponding to these four applications could not be generated even after multiple attempts at debugging. The outputs for these applications are present, however, for the Questions 3 and 4 as mentioned later in the report.

Question 3

Implementation Details

File Convention
The code for this question is contained in the file task_3.cpp

Instruction Routine

For Memory Trace, use the IARG_MEMORYOP_EA parameter provided by PIN to pass the Memory Operand address
Call the analysis routine separately for read and write instructions

```
UINT32 memOperands = INS_MemoryOperandCount(ins);

// Iterate over each memory operand of the instruction.
for (UINT32 memOp = 0; memOp < memOperands; memOp++)
{
    double refSize = INS_MemoryOperandSize(ins, memOp);
    if (INS_MemoryOperandIsRead(ins, memOp))
    {
        INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
        INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)MemoryTrace, IARG_PTR, &refSize, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
    }
    if (INS_MemoryOperandIsWritten(ins, memOp))
    {
        INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
        INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)MemoryTrace, IARG_PTR, &refSize, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
    }
}
```

For Instruction Trace, use the IARG_INST_PTR to pass the instruction address being accessed

```
int ins_size = INS_Size(ins);

INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)InstructionTrace, IARG_PTR, &ins_size, IARG_INST_PTR, IARG_END);

INS_InsertCall(ins, IPOINT_BEFORE, InsCount, IARG_END);
```

Analysis Routines

- MemoryTrace(): Takes in the pointer to the memory being accessed, extracts the address in the numerical form by typecasting, divides the address by the granularity (32 in our case), and then multiplies back by granularity value to get the memory word being accessed. Finally, stores it in a set.
- InstructionTrace(): Similar logic as MemoryTrace(), only difference is that it takes the corresponding instruction address and stores it in the corresponding set.

- ExitRoutine(): Prints the number of unique elements in both the above sets, representing the number of unique memory addresses and instruction addresses accessed respectively.

```
// Insert the Memory Address acccessed
VOID MemoryTrace(VOID* sz, VOID* addr) {
    mem_st.insert(((*(double*)addr)/GRANULARITY)*GRANULARITY);
}

// Insert the Instruction Adddress accessed
VOID InstructionTrace(VOID* sz, VOID* addr) {
    ins_st.insert(((*(double*)addr)/GRANULARITY)*GRANULARITY);
}
```

Results

Application	Unique Memory Addresses Accessed	Unique Instruction Addresses Accessed
400.perlbench	10497	539
401.bzip2	580270	157
403.gcc	40340	74
429.mcf	2130790	39
450.soplex	84781	204
456.hmmer	4395	157
471.omnetpp	29927	236

Question 4

Implementation Details

File Convention
The code for this question is in the file task_4.cpp

Data Structures

Vectors of appropriate size are used for keeping track of each of the counts mentioned in the question. For example, ins_len[len] stores the number of instructions having length len, op_count[cnt] stores the number of instructions having operand count as cnt, and so on.

(Here, MAXSIZE = 25 since x86 has maximum instruction size = 17)

```
vector<long double>  ins_len(MAXSIZE, 0);
vector<long double> op_count(MAXSIZE, 0);
vector<long double>  reg_read_ops(MAXSIZE, 0);
vector<long double>  reg_write_ops(MAXSIZE, 0);
vector<long double> mem_op_counts(MAXSIZE, 0);
vector<long double>  mem_read_counts(MAXSIZE, 0);
vector<long double> mem_write_counts(MAXSIZE, 0);
vector<long double> mem_accesses(ACCESSVECTOR, 0);
```

Instrumentation Routine

The instrumentation routine code for Tasks (a)-(d) are as shown in the snapshot below. The pointer to the corresponding vector index is passed to the corresponding analysis routine, which then simply increments the count stored at this address.

```
// Task (a)
long double ins_sz = INS_Size(ins);

INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)Ins_lengths, IARG_PTR, &(ins_len[ins_sz]), IARG_END);

// Task (b)
long double opcount = INS_OperandCount(ins);
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)Op_counts, IARG_PTR, &(op_count[opcount]), IARG_END);

// Task (c)
long double regr_op_count = INS_MaxNumRRegs(ins);
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)Regr_ops, IARG_PTR, &(reg_read_ops[regr_op_count]), IARG_END);

// Task (d)
long double regw_op_count = INS_MaxNumWRegs(ins);
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR) FastForward, IARG_END);
INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)Regw_ops, IARG_PTR, &(reg_write_ops[regw_op_count]), IARG_END);
```

The instrumentation routine code for tasks (e)-(h) are shown in the snapshot below. In these tasks, only predicated instructions are considered as mentioned in the question. The logic is same as tasks (a)-(d) as the pointer corresponding to the counter to be incremented is passed to the corresponding analysis routine, which then simply increments the counter value.

```
// Task (e)
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)FastForward, IARG_END);
INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)MemOp_counts, IARG_PTR, &(mem_op_counts[memOp_count]), IARG_END);

// Task (f)
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)FastForward, IARG_END);
INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)MemRead_counts, IARG_PTR, &(mem_read_counts[memRead_count]), IARG_END);

// Task (g)
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)FastForward, IARG_END);
INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)MemWrite_counts, IARG_PTR, &(mem_write_counts[memWrite_count]), IARG_END);

// Task (h)
INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)FastForward, IARG_END);
INS_InsertThenPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)Mem_access, IARG_PTR, &(mem_accesses[mem_access]), IARG_END);
```

Analysis Routine

The analysis routine consists simply of 8 functions responsible for incrementing the counters corresponding to the tasks (a)-(h)

Results

400.perlbench

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	1.17×10^8
2	2.56×10^8
3	2.75×10^8
4	5.29×10^7
5	7.85×10^7
6	1.85×10^8
7	3.39×10^7
8	28
9	0
10	12

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	972133
1	1.075×10^6
2	5.2×10^8
3	3.55×10^8
4	1.03×10^8
5	1.56×10^7
6	3.25×10^6

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions

0	1.02×10^7
1	2.6×10^8
2	5.38×10^8
3	1.79×10^8
4	7.02×10^6
5	2.57×10^6
6	3.25×10^6

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	1.36×10^8
1	6.85×10^8
2	1.75×10^8
3	2.38×10^6
4	873273

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	4.53×10^8
1	5.31×10^8
2	1.55×10^7

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	**Number of Instructions**
0	6.44×10^8

1	3.54×10^8
2	872350

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	**Number of Instructions**
0	7.94×10^8
1	2.05×10^8

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 2.04

403.gcc

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	1.82×10^8
2	3.05×10^8
3	2.82×10^8
4	8.76×10^7
5	5.34×10^7
6	7.62×10^7
7	1.24×10^7

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	3.91×10^6
1	2.49×10^6
2	4.26×10^8
3	3.63×10^8
4	1.73×10^8
5	3.06×10^7

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions
0	7.27×10^6
1	3.17×10^8
2	4.82×10^8
3	1.89×10^8

4	3.57×10^6
5	401409

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	6.75×10^7
1	7.24×10^8
2	2.07×10^6

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	4.95×10^8
1	4.98×10^8
2	5.16×10^6

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	Number of Instructions
0	6.61×10^8
1	3.38×10^8

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	8.29×10^8
1	1.70×10^8

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 1.9

429.mcf

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	8.062×10^7
2	4.854×10^8
3	3.155×10^8
4	5.053×10^7
5	2.207×10^7
6	5.249×10^6
7	4.059×10^7

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	1.477×10^6
1	0
2	4.848×10^8
3	4.573×10^8
4	4.383×10^7
5	1.255×10^7

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions
0	3.76×10^6
1	1.485×10^8
2	6.775×10^8
3	1.702×10^8

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	7.09×10^7
1	7.702×10^8
2	1.587×10^8
3	38512

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	4.87×10^8
1	5×10^8
2	1.243×10^7

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	Number of Instructions
0	5.847×10^8
1	4.15×10^8

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	8.8996×10^8
1	1.1×10^8

- Maximum size (in Bytes) of memory accesses across all instructions: 4
- Average size (in Bytes) of memory accesses across all instructions: 2.051

450.soplex

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
4	1.6×10^7
6	4×10^7
7	1.77×10^7
8	3.93×10^6

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	3550
1	13
2	4.118×10^8
3	3.963×10^8
4	1.886×10^8
5	3.21×10^6

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions
0	2.26×10^7
1	2.16×10^8
2	5.79×10^8
3	1.36×10^8
4	4.5×10^7
5	31137

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	6.77×10^7
1	7.644×10^8
2	1.677×10^8

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	5.31×10^8
1	4.4×10^8
2	2.78×10^7

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	Number of Instructions
0	5.83×10^8
1	4.16×10^8

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	9.2×10^8
1	7.96×10^7

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 2.35

456.hmmer

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	2.51×10^7
2	3.03×10^8
3	2.96×10^8
4	2.7×10^8
5	2.48×10^7
6	6.83×10^7
7	416480
8	1.18×10^7

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	34887
1	2512
2	5.66×10^8
3	4.33×10^8
4	713115
5	159551
6	23808

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions
0	610947
1	7.56×10^7
2	5.62×10^8
3	2.81×10^8

4	7.99×10^7
5	14929
6	23808

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	7.51×10^7
1	7.55×10^8
2	1.69×10^8
3	23808

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	Number of Instructions
0	3.76×10^8
1	6.23×10^8
2	166109

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	Number of Instructions
0	4.52×10^8
1	5.47×10^8

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	9.24×10^8
1	7.56×10^7

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 2.49

471.omnetpp

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	1.54×10^8
2	3.08×10^8
3	3.82×10^8
4	3.43×10^7
5	4.511×10^7
6	4.88×10^7
7	2.65×10^7
8	0
9	0
10	421

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	802402
1	225879
2	5.182×10^8
3	2.817×10^8
4	1.727×10^8
5	2.5×10^7
6	1.13×10^6

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions

0	2.9×10^7
1	1.975×10^8
2	5.4×10^8
3	2.197×10^8
4	8.475×10^6
5	3.68×10^6
6	1.13×10^6

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	1.65×10^8
1	6.65×10^8
2	1.67×10^8
3	376014
4	1.13×10^6

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	4.49×10^8
1	5.43×10^8
2	7.76×10^6

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	**Number of Instructions**
0	6.61×10^8
1	3.37×10^8

2	1.13×10^6

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	7.81×10^8
1	2.19×10^8

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 2.32

483.xalancbmk

- Distribution of Instruction Lengths

Instruction Length	Number of Instructions
1	1.44×10^8
2	3.32×10^8
3	4.43×10^8
4	2.91×10^7
5	2.19×10^7
6	2.17×10^7
7	6.94×10^6
8	233987
9	54843
10	59729

- Distribution of Number of Operands in an Instruction

Number of Operands	Number of Instructions
0	2.53×10^7
1	672226
2	4.1×10^8
3	4.27×10^8
4	9.39×10^7
5	2.25×10^7
6	1.92×10^7

- Distribution of Number of Register Read Operands in an Instruction

Number of Register Read Operands	Number of Instructions

0	3.24×10^7
1	2.43×10^8
2	4.45×10^8
3	2.49×10^8
4	1.09×10^6
5	8.81×10^6
6	1.92×10^7

- Distribution of Number of Register Write Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	1.09×10^8
1	6.9×10^8
2	1.8×10^8
3	1.92×10^7

- Distribution of Number of Memory Operands in an Instruction

Number of Register Write Operands	**Number of Instructions**
0	5.11×10^8
1	4.58×10^8
2	3.05×10^7

- Distribution of Number of Memory Read Operands in an Instruction

Number of Memory Read Operands	**Number of Instructions**
0	6.41×10^8
1	3.58×10^8

- Distribution of Number of Memory Write Operands in an Instruction

Number of Memory Write Operands	Number of Instructions
0	8.39×10^8
1	1.6×10^8

- Maximum size (in Bytes) of memory accesses across all instructions: 8
- Average size (in Bytes) of memory accesses across all instructions: 2.04