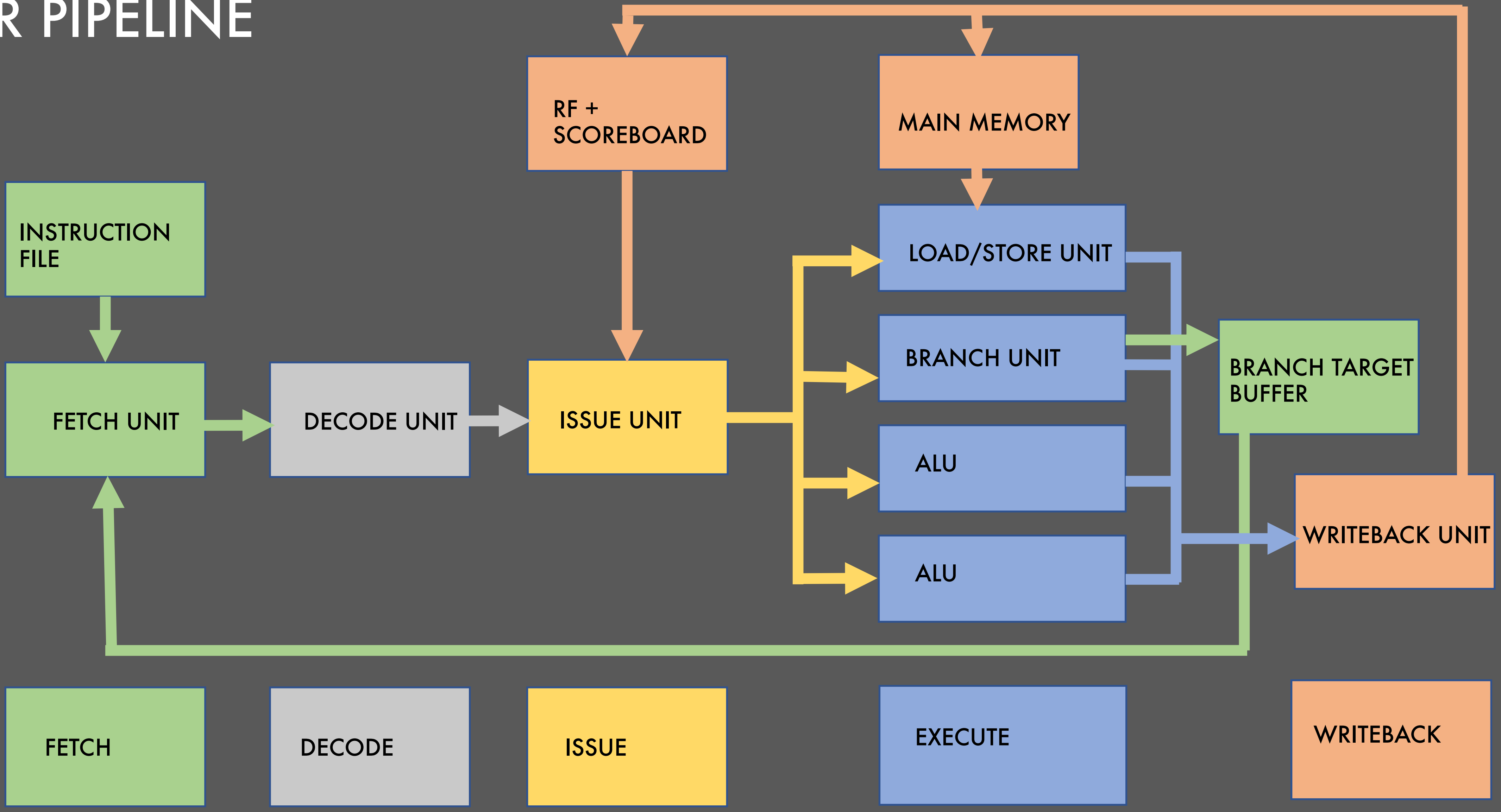
ADVANCED COMPUTER ARCHITECTURE : PROCESSOR SIMULATOR

Joshua Measure-Hughes

Student number: 1744589

PROCESSOR ARCHITECTURE PER PIPELINE



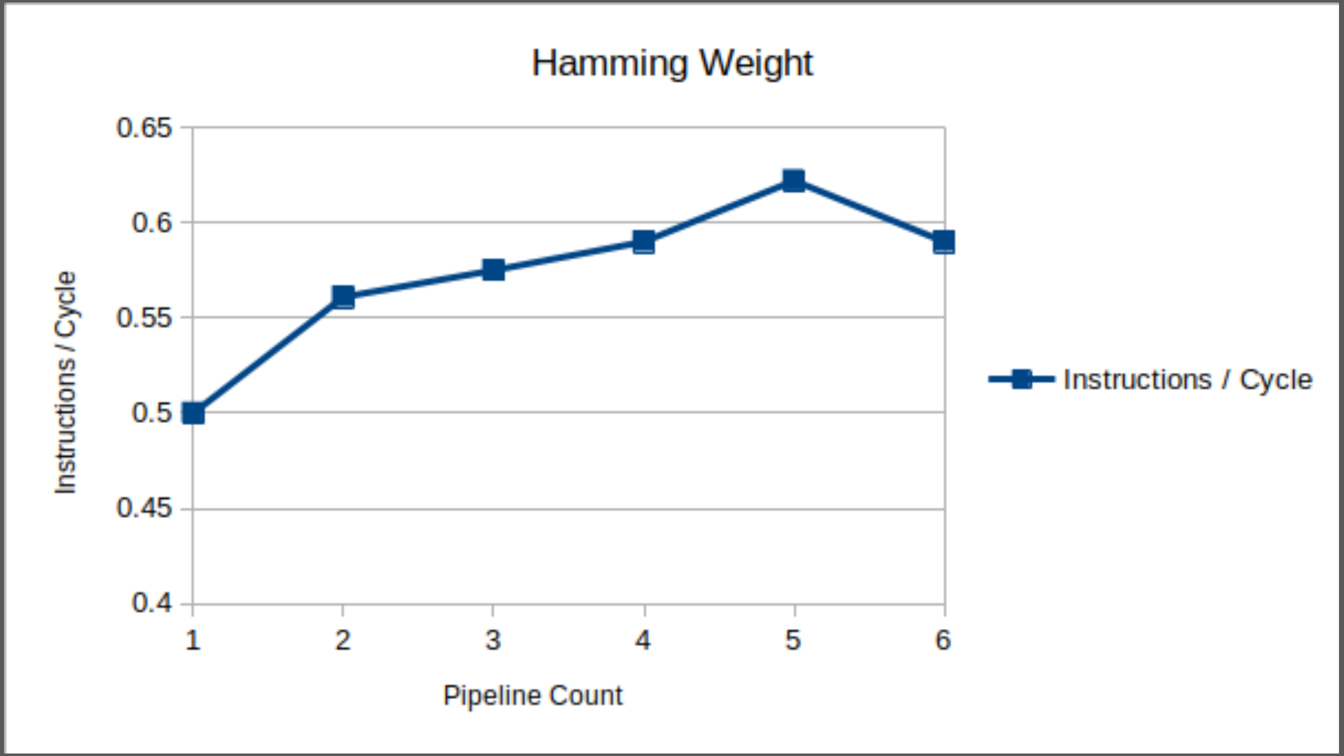
ARCHITECTURAL and MICRO - ARCHITECTURAL FEATURES

- Pipelined 5-stage: fetch, decode, issue, execute, write-back
- Per pipeline: 2 ALUs, Branch Unit, Load/Store Unit
- Configurable number of pipelines
- Reorders Issue Units based on age of stalled instruction
- Out of Order execution using Register File scoreboard
- Branch predictor is configurable
  - Static
  - Fixed
  - 1-bit dynamic
  - 2-bit dynamic
  - False predictions flush the pipeline, fetch starts at correct PC on the next cycle, Branch Target Buffer updated with true branch result

INSTRUCTION SET

Instruction Format	Description	Verbose Description
`add rd, rs1, rs2`	Add	`rd ← rs1 + rs2, pc ← pc + INC`
`sub rd, rs1, rs2`	Subtract	`rd ← rs1 - rs2, pc ← pc + INC`
`mul rd, rs1, rs2`	Multiply	`rd ← rs1 * rs2, pc ← pc + INC`
`and rd, rs1, rs2`	AND	`rd ← rs1 & rs2, pc ← pc + INC`
`or rd, rs1, rs2`	OR	`rd ← rs1 | rs2, pc ← pc + INC`
`xor rd, rs1, rs2`	XOR	`rd ← rs1 ^ rs2, pc ← pc + INC`
`beq rs1, rs2, pcrel`	Branch Equal	`pc ← pc + ((rs1==rs2) ? pcrel : INC)`
`bnq rs1, rs2, pcrel`	Branch Equal	`pc ← pc + ((rs1!=rs2) ? pcrel : INC)`
`jmp rd, pcrel`	Jump	`rd ← pc + 4, pc ← pc + pcrel`
`ld rd, rs1`	Load	`rd ← mem[rs1], pc ← pc + INC`
`ldi rd, imm`	Load Immediate	`rd ← imm, pc ← pc + INC`
`lds rd, rs2, rs2`	Load Scaled	`rd ← mem[rs1 + rs2 * d], pc ← pc + INC`
`st rs1, rd`	Store	`mem[rd] ← rs1, pc ← pc + INC`
`cmp rs1, rs2, rd`	Compare	`rd ← (rs1 > rs2) ? 1 : (rs1 < rs2 ? -1 : 0), pc ← pc + INC`
`shr rs1, rs2, rd`	Shift Right	`rd ← (rs1 >> rs2), pc ← pc + INC`
`shl rs1, rs2, rd`	Shift Left	`rd ← (rs1 << rs2), pc ← pc + INC`
`hlt`	Halt	`stop execution`

EXPERIMENT 3: HAMMING WEIGHT

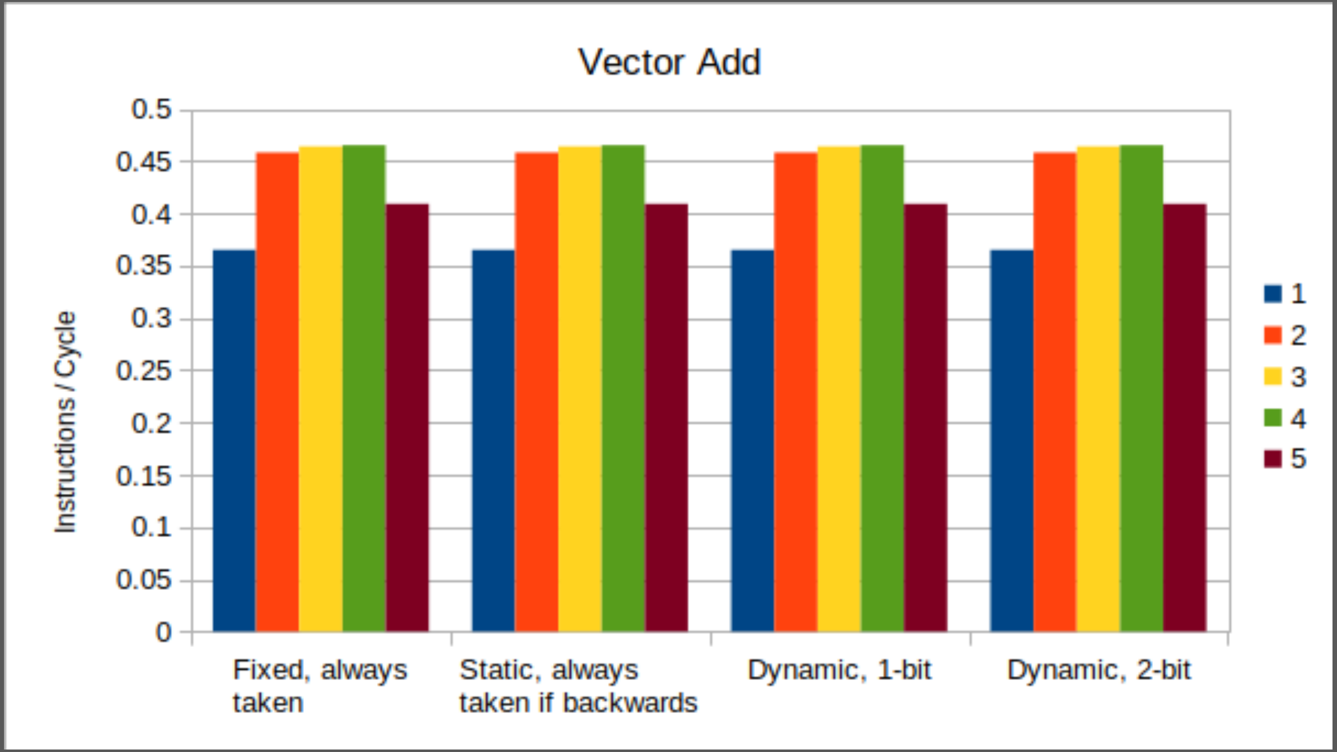


HYPOTHESIS: Increasing the number of pipelines will increase or not change the instructions / cycle of hamming weight.

RESULT: The instructions / cycle increased initially alongside pipeline count but dropped back down at 6 pipelines.

EVALUATION: This is clearly too many pipelines for the Hamming Weight benchmark. The results can be explained by the fact that there are data dependencies within the benchmark, and these resulted in a lot of stalls within the pipelines when the pipeline count was too large, reducing the instructions / cycle count.

EXPERIMENT 3: VECTOR ADDITION

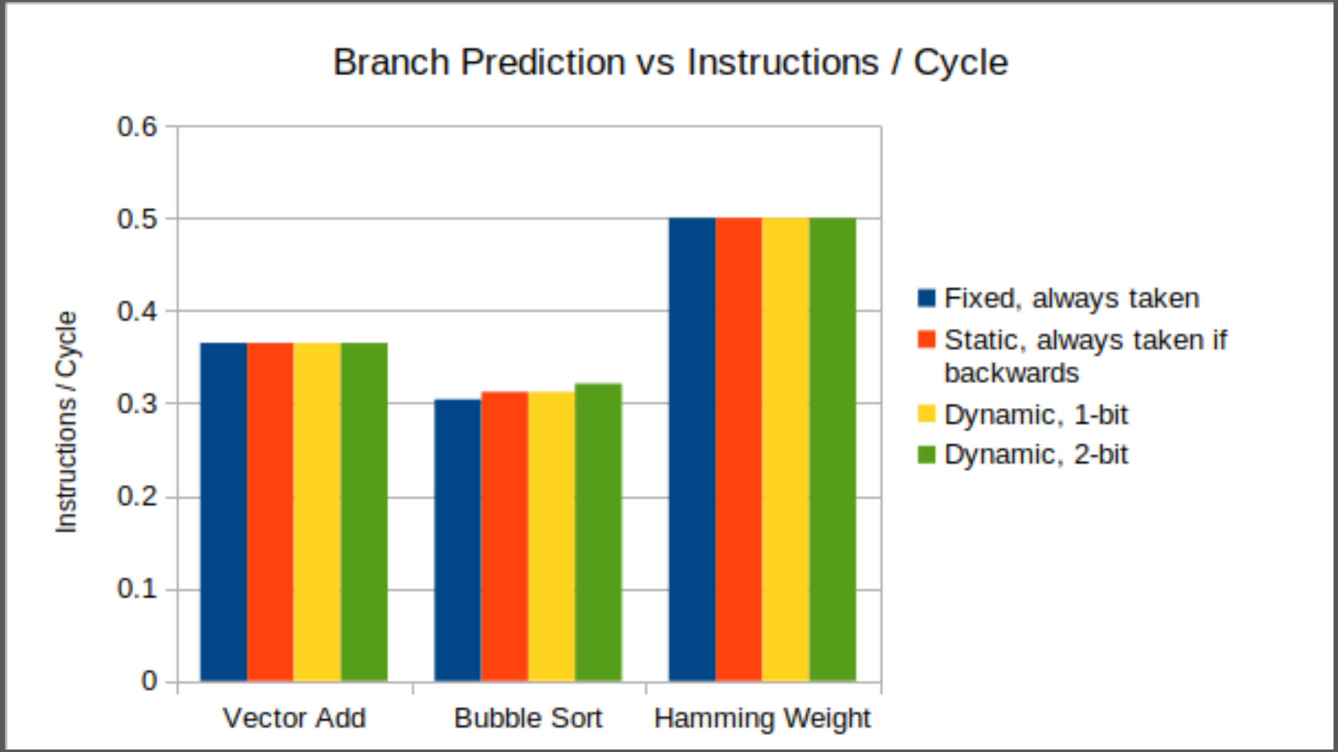


HYPOTHESIS: The number of pipelines will always increase the instructions / cycle, regardless of branch predictor type.

RESULT: The instructions / cycle increased with all branch prediction types until it an optimum, and for vector add this was 4 pipelines.

EVALUATION: As vector add has many arithmetic operations, it makes sense that the increased pipeline count and thus ALU count improved the instructions / cycle initially, regardless of the branch predictor type. The threshold of 4 pipelines also makes sense, as there are roughly 4 instructions between branches, although I would have expected the instructions / cycle count for the 5th pipeline to be higher for the dynamic branch predictors. There are quite a few false dependencies within the code, and as there was no register renaming, it was expected that the instructions /cycle was not higher.

EXPERIMENT 3: BRANCH PREDICTION



HYPOTHESIS: The instructions / cycle rate will increase or not change with complexity of branch predictor, regardless of benchmark.

RESULT: The instructions / cycle remained unchanged for vector add and hamming weight, and improved for bubble sort.

EVALUATION: The vector add contains a lot of backwards branches and so a fixed, static, 1-bit or 2-bit predictor are likely to produce the same results, as seen in the graph.
The hamming weight benchmark contained no branches at all, and so the fixed result makes perfect sense. The bubble sort contains a mixture of backwards branches, forward branches, and jumps, and so a dynamic branch predictor was much more suited to the program.