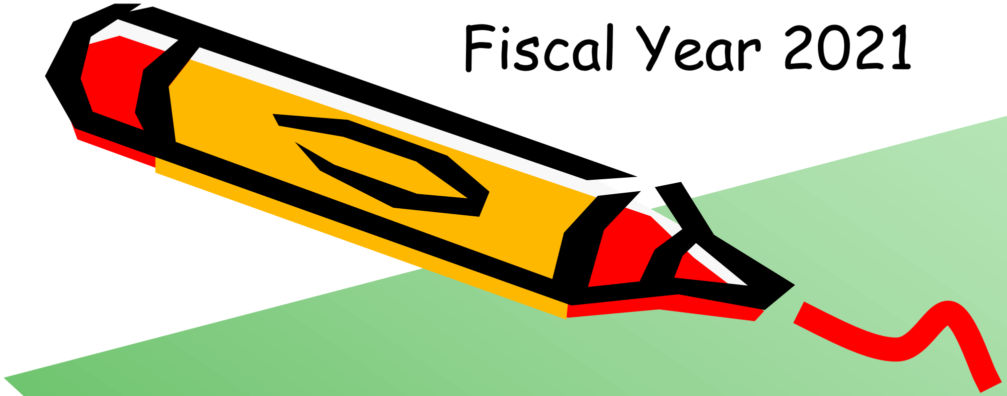


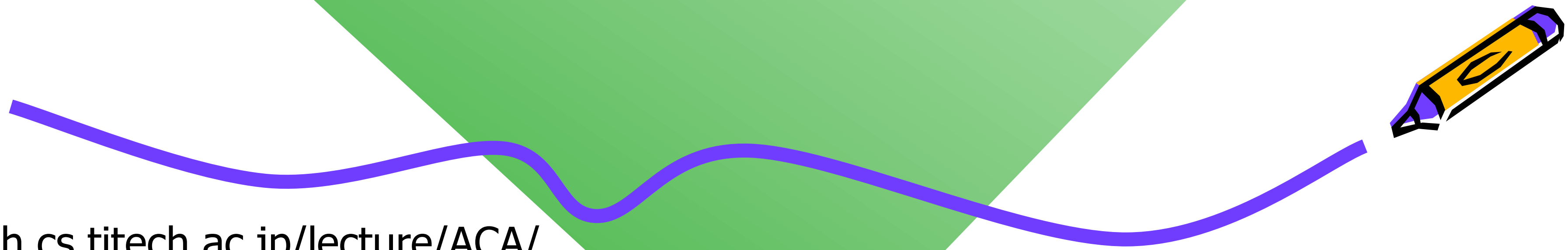
Fiscal Year 2021

Ver. 2022-02-08a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

14. Final report



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 14:20-16:00, Thr 14:20-16:00

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Final report

1. Submit your final report describing your answers to questions 1 - 7 in a PDF file
   via E-mail by February 17, 2022
2. Enjoy!

1. Academic paper reading

- Select an academic paper from the list below and
  - In your own word, describe the problem that the authors try to solve,
  - Describe the key idea of the proposal,
  - Describe your opinion why the authors could solve the problem although there may be many researchers try to solve similar problems.
- List
  - Prophet/critic hybrid branch prediction, ISCA'04, 2004
  - The V-Way Cache: Demand Based Associativity via Global Replacement, ISCA'05, 2005
  - Emulating Optimal Replacement with a Shepherd Cache, MICRO-40, 2008
  - A new case for the TAGE branch predictor, MICRO-44, 2011
  - Skewed Compressed Caches, MICRO-47, 2014
  - Focused Value Prediction, ISCA, 2020

2. MIPS assembly programming

- Write MIPS assembly code asm1.s for code1.c in C.

```
int sum = 0;
int i, j;
for (i=0; i=<100; i++)
  for (j=0; j=<100; j++) sum += (j+i);
```

code1.c

- Write MIPS assembly code asm2.s for code2.c in C.

```
int A[200];
int sum = 0;
int i;
for (i=0; i<200; i++) A[i] = i;              /* initialize the array */
for (i=1; i<200; i++) A[i] = A[i-1] + A[i]; /* compute              */
for (i=0; i<200; i++) sum += A[i];           /* obtain the sum       */
```

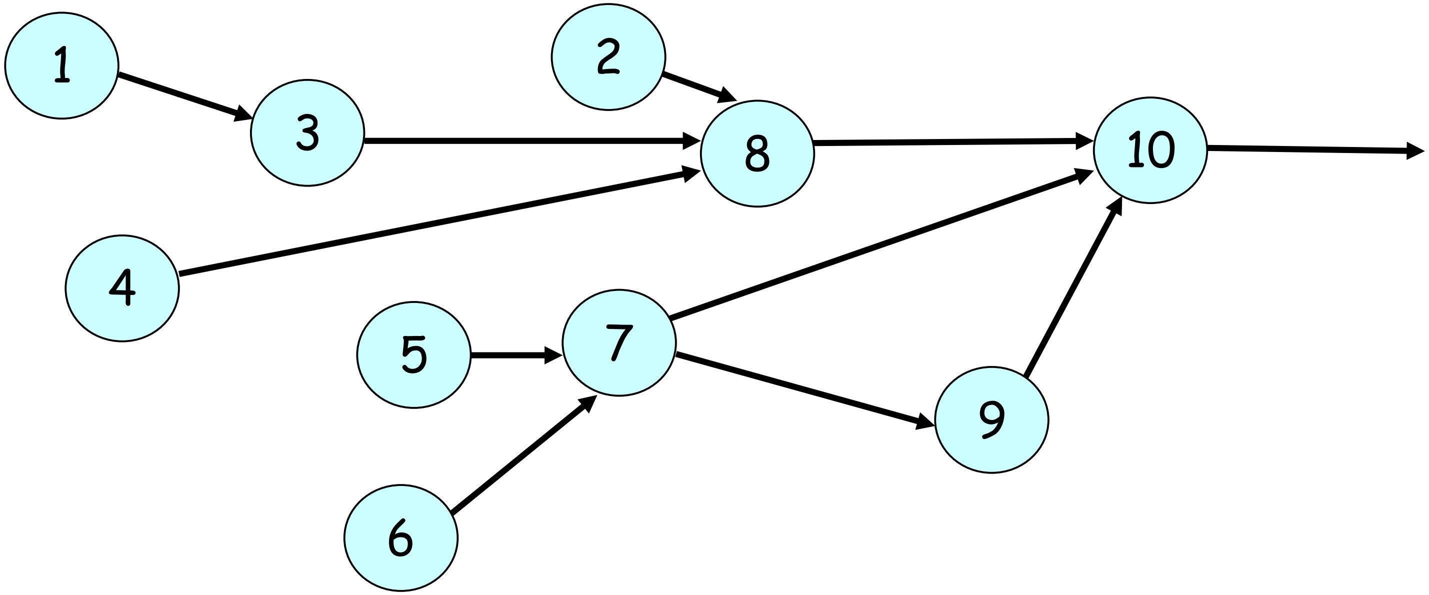
code2.c

3. Pipelined processor

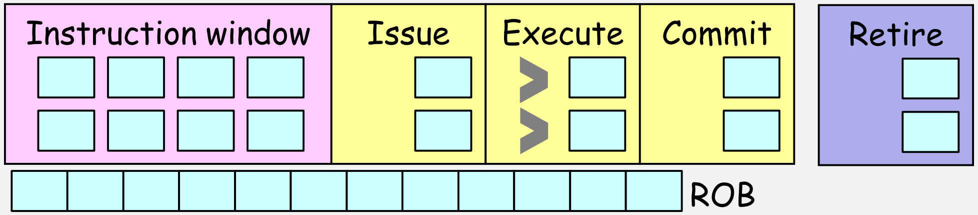
- Design a five stage pipelined scalar processor supporting MIPS add, addi, lw, sw, and bne instructions in Verilog HDL. Please download proc08.v from the support page and refer it.
  Note that you do not need to implement data forwarding.
- Verify the behavior of designed processor using asm1.s and asm2.s.
  You may insert NOP instructions if necessary.
- The report should include a block diagram, a source code in Verilog HDL, the description of the changes of the code, and obtained waveforms of your design.

4. OoO execution and dynamic scheduling

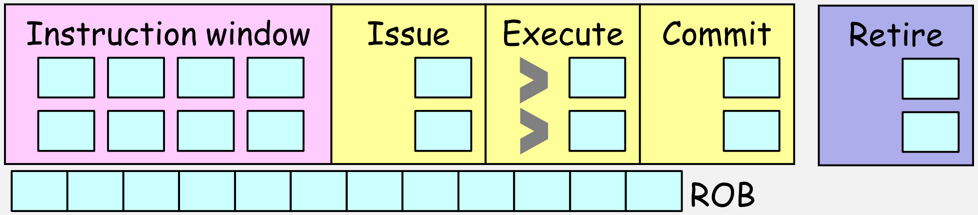
- Draw the cycle by cycle processing behavior of these 10 instructions
- Modify this dataflow graph and draw another cycle by cycle processing behavior of the graph having 10 instructions



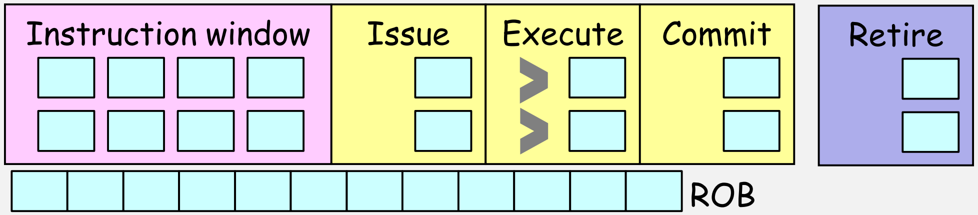
Cycle 1



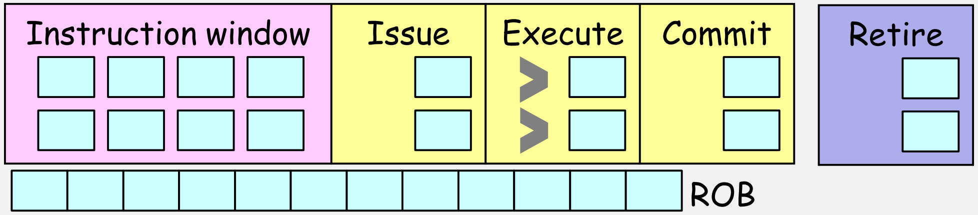
Cycle 2



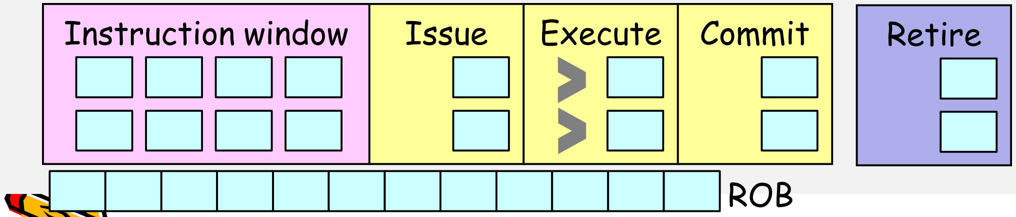
Cycle 3



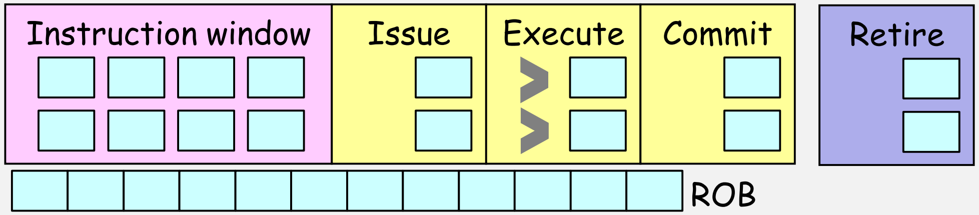
Cycle 4



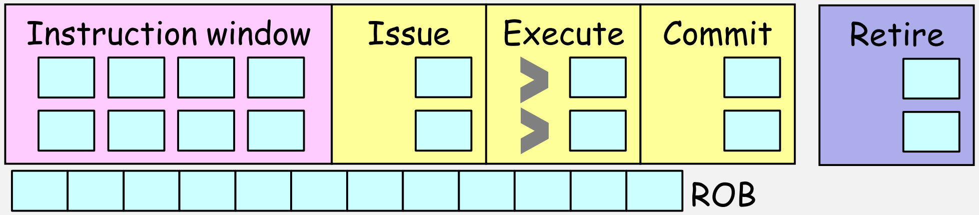
Cycle 5



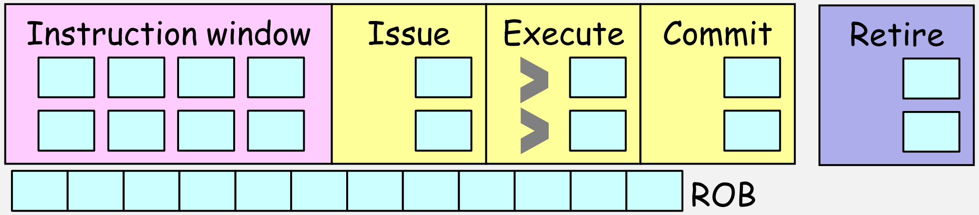
Cycle 6



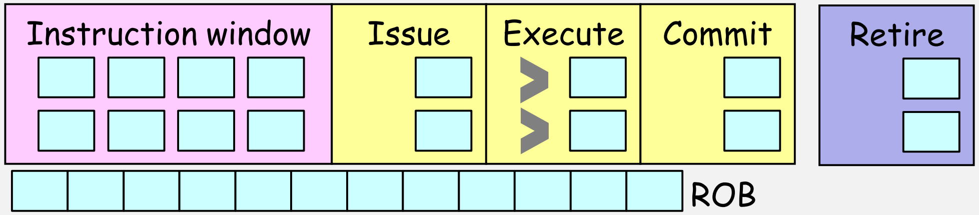
Cycle 7



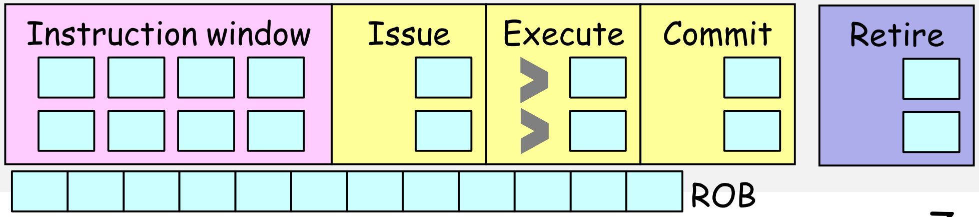
Cycle 8



Cycle 9



Cycle 10



5. Parallel programming (The free lunch is over)

- Describe an efficient parallel program for the following sequential program using LOCK(), UNLOCK() and BARRIER() assuming a shared memory architecture
- Explain why your cord runs correctly and why your code is efficient.

main02.c

```c
#define N 8      /* the number of grids */
#define TOL 15.0 /* tolerance parameter */
float A[N+2], B[N+2];

void solve () {
    int i, done = 0;
    while (!done) {
        float diff = 0;
        for (i=1; i<=N; i++) { /* use A as input */
            B[i] = 0.333 * (A[i-1] + A[i] + A[i+1]);
        }
        for (i=1; i<=N; i++) { /* use B as input */
            A[i] = 0.333 * (B[i-1] + B[i] + B[i+1]);
            diff = diff + fabsf(B[i] - A[i]);
        }
        if (diff <TOL) done = 1;
        for (i=0; i<=N+1; i++) printf("%6.2f ", B[i]);
        printf("| diff=%6.2f¥n", diff); /* for debug */
    }
}

int main() {
    int i;
    for (i=1; i<N-1; i++) A[i] = B[i] = 100+i*i;
    solve();
}
```

6. Building blocks for synchronization

- Fetch-and-increment reads an original value from memory and increments (adds one to) it in memory atomically
- Implement fetch-and-increment (FAI) using the load-linked/store-conditional instruction pair
    - Refer the discussion of implementing an atomic exchange EXCH
- Implement BARRIER() using FAI

7. Cache coherence protocols

- Select your favorite commercial multi-core processor
  - Describe the memory organization including caches and main memory
    - cache line size, write policy, write allocate/no-allocate, direct-mapped/set-associative, the number of caches (L1, L2, and L3?)
  - Describe the cache coherence protocol used there