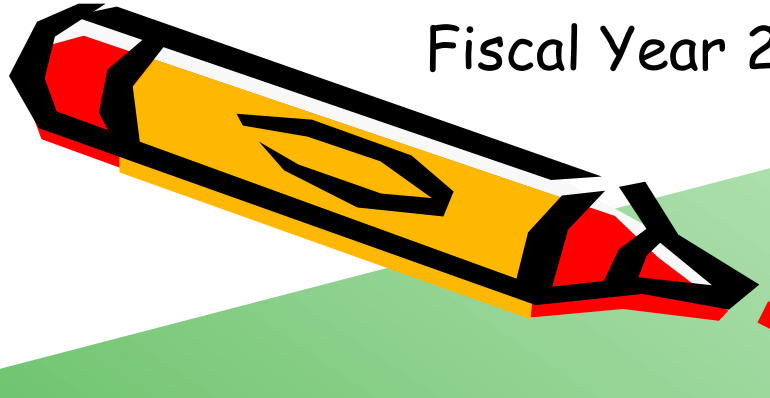


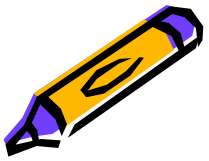
Fiscal Year 2023

Ver. 2024-01-25a

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

Advanced Computer Architecture

Final Report



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W834, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

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

Final report of Advanced Computer Architecture

1. Please submit your final report describing your answers to questions 1 - 7 in a PDF file
   via E-mail (kise [at] c.titech.ac.jp) by February 13, 2024
   - E-mail title should be "Report of Advanced Computer Architecture"
2. Please submit the report in 16 pages or less on A4 size PDF file, including the cover page.

3. 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, 2004
  - Focused Value Prediction, ISCA, 2020
  - Clockhands: Rename-free Instruction Set Architecture for Out-of-order Processors, MICRO, 2023
  - Emulating Optimal Replacement with a Shepherd Cache, MICRO, 2008

2. RISC-V assembly programming

- Write RISC-V assembly code asm1.s for code1.c in C. Use Venus RISC-V editor and simulator to show that the output of the code you wrote is correct.

```
int sum = 0;
int i, j;
for (i=1; i=<100; i=i+2)
  for (j=1; j=<100; j++) sum += (j+i);
```

code1.c

- Write RISC-V assembly code asm2.s for code2.c in C. Use Venus RISC-V editor and simulator to show that the output of the code you wrote is correct.

```
int A[200];
int sum = 0;
int i;
for (i=0; i<200; i++) A[i] = 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       */
```

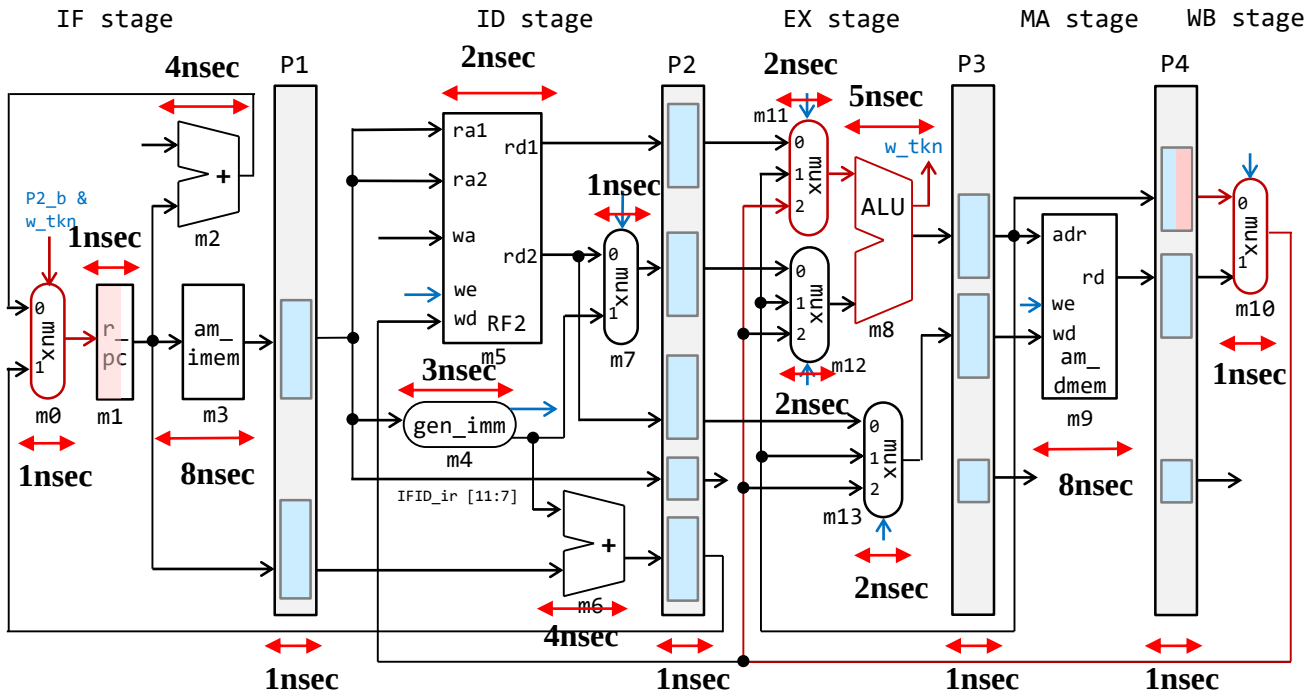
code2.c

3. Pipelined processor

- Design a 3-stage pipelined scalar processor supporting RISC-V add, addi, lw, sw, and bne instructions.
  Configure the critical paths of the three stages to have smaller delay assuming each module delay of the next slide
- The report should include a block diagram and the description of the changes of your design.
- Show the delay of the critical path of your design.

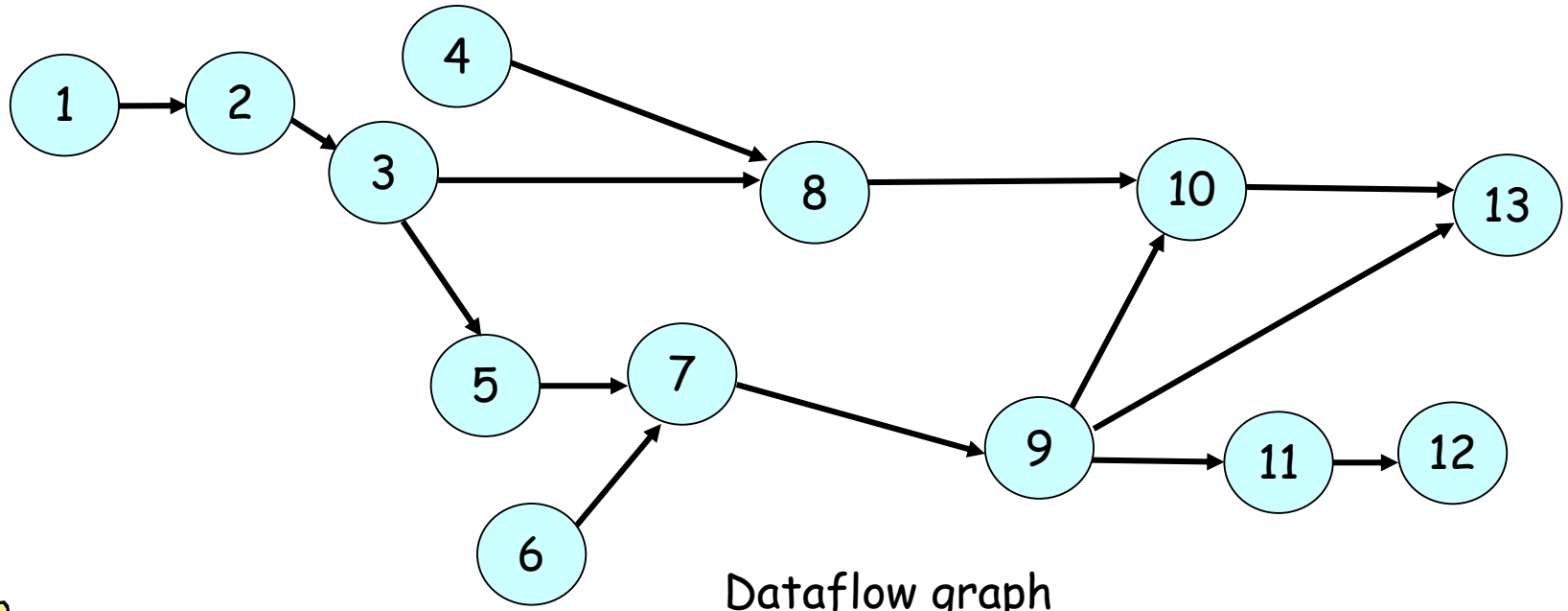
proc9: critical path of 5-stage pipelining processor

- The path from P4 pipeline register to PC is the critical path.
- This 5-stage organization is commonly explained in typical computer architecture textbooks.



4. OoO execution and dynamic scheduling

- Draw the cycle by cycle processing behavior of these 13 instructions
- Modify this dataflow graph by removing two edges of the graph so that the number of execution cycles is reduced. Draw another cycle by cycle processing behavior of the modified graph.



Dataflow graph


Cycle 1
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 2
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 3
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 4
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 5
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 6
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 7
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 8
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 9
Instruction window
Issue
Execute
Commit
Retire
ROB
Cycle 10
Instruction window
Issue
Execute
Commit
Retire
ROB

5. Parallel programming

- Adjust the number of elements N so that this sequential program (main7.c) takes about 1 second. Use this adjusted value for N. Use the `time` command to measure the execution time.
- Describe an efficient parallel program for the sequential program of main7.c using LOCK, UNLOCK, and BARRIER of pthread assuming a shared memory architecture of 4 cores.
- Explain why your code runs correctly and why your code is efficient.
- Show your speedup over the sequential execution. To measure the execution time, use a computer with four or more cores.

```
#include <stdio.h>
#include <math.h>
#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;
    float diff;
    while (!done) {
        diff = 0;
        for (i=1; i<=N; i++) {
            B[i] = 0.333 * (A[i-1] + A[i] + A[i+1]);
            diff = diff + fabsf(B[i] - A[i]);
        }
        if (diff <TOL) done = 1;
        for (i=1; i<=N; i++) A[i] = B[i];
    }
    printf("diff=%6.2f¥n", diff);
}

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

main7.c

6. Building blocks for synchronization

- Implement your BARRIER() using some global variables, pthred lock, and unlock.
- Show your code and explain why your code runs correctly and why your code is efficient.

- Replace the barrier in the program in Question 5 with your designed one and measure the speedup over the sequential program.

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