# Control-Data Decoupled Architecture

Daniel Nemirovsky, Nikola Markovic, Osman Unsal
Adrian Cristal, Mateo Valero

## Motivation

- **Problem**: Extracting parallelism from high level languages to run on CMPs without affecting the programmer's productivity
- **Solution**: Using Object Oriented techniques, we propose a decoupled hw architecture that separates a program's control and data flow
  - Parallelization through the layers

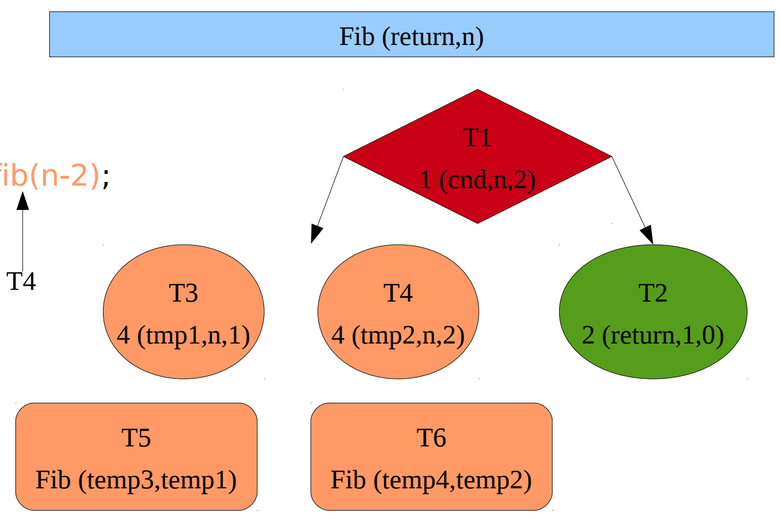
## Initial Work

- Single Core with 3 layers
  - Control Processor – sequentially executes program workload
  - Data Control Engine – uses mirrored register window mechanism, has similar functionalities as DMA
  - Data Processor – simple scalar core (MIPS)
- Hand compiled set of initial microbenchmarks
- Measurements and analysis of model

## Fine Grain Example

```
Int fib(int n) {
   if (n<=1) return 1;
   else return fib(n-1) + fib(n-2);
}
```

T7 T5 T3 T6 T4



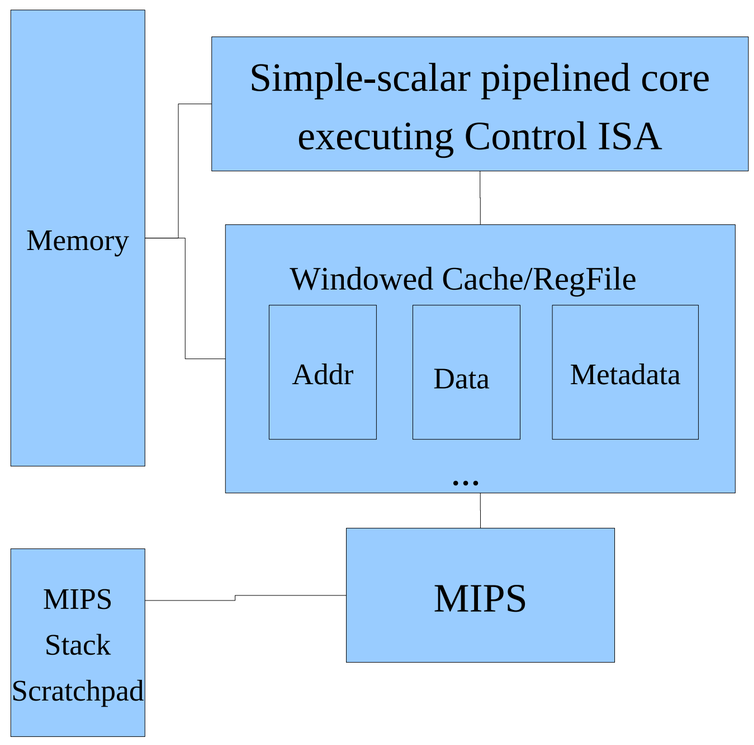

T3:
1. create T3 [4] [3 args]
2. send T3 temp1 [0]
3. send T3 n [1]
4. send T3 1 [2]
5. call T3

MIPS basic type Int tasks:
1. slti $3, $1, $2
2. sw $2, $1, offset
3. add $3, $1, $2
4. sub $3, $1, $2

## Control Flow ISA

- Create – requests memory allocation and reserves space for new object/method to be created
- Send – sends needed references and arguments to methods
- Call – triggers the execution of methods
- Return – returns to parent method at correct PC

## Microarchitecture




## Parallelization Problems

- Conditions
  - Conditions are handles as basic tasks and all control instructions are predicated
- Dependencies
  - Area of data flow model brainstorming (versioning, shared data mapping)

## Future Work and Possible Advantages

- Software
  - Easy compilers
  - Improvements are done at runtime
- Hardware
  - Better approaches for data and execution locality
  - Asynchronous and speculative execution
- Execution Model
  - Runtime decisions are taken by hardware
  - Variable levels of parallelism
  - Distributed Execution Model