Formal description of creating a central controller

Zdeněk Přikryl
TID, 2007

Content

- Introduction
- What is a “Central controller”
- Language ISAC
- Creation of the central controller from the language ISAC
- Conclusion

Introduction

- Vendors of SoCs and embedded systems demand new processors in very short time
- One needs reliable tool for fast description and creation processors
- Developers wants to describe the first prototype of processor in some high level abstraction language
  - after successful instruction accurate simulations, developers take care about HW realization and performs cycle accurate simulations
- Even for an instruction accurate simulation one needs some controller and decoder
  - we will focus on the central controller

Central controller 1/2

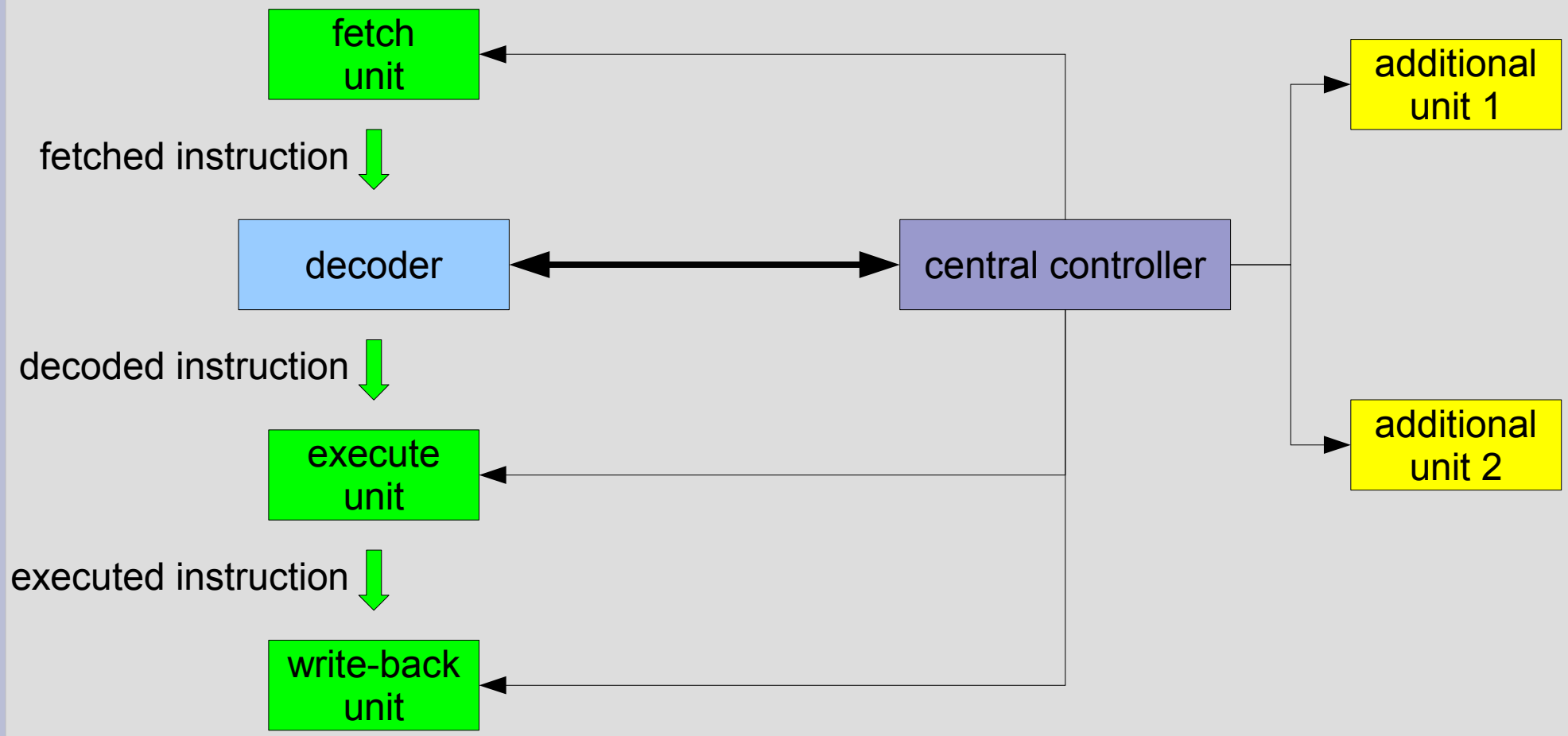
- A controller is a device which monitors and affects the operational conditions of a given dynamical system. The operational conditions are typically referred to as output variables of the system which can be affected by adjusting certain input variables.
  - **distributed controller** is divided in separate functional units which are placed in different parts on a chip and communicate between each other by signals
  - **central controller** is one functional unit which is placed somewhere in a chip

Central controller 2/2

- The controller needs information about what will he has to do in future
  - for example:
    - activate unit 1 on every tenth clock cycle
    - activate decode unit on next clock cycle
    - ...
- Usually this information is stored in only one state variable

- A Decoder is a device which accepts an encoded instruction in a byte code and try to decode the sense
  - decoder can affect the controller

Illustration



Language ISAC

- High level abstraction language developed for the project LISSOM
  - language is inspired by the language LISA
- Two basic constructions for instruction set
  - **Operation** – denotes an instruction in assembly language (Assembler section) and machine language (Coding section)
  - **Group** – aggregates similar operations
- Four basic constructions for timing model
  - **Activation** – when and what will be done
  - **Coding Root** – when and what will be decoded
  - **Pipe Line**
  - **%** *number* – explicit delay
    - *when* is identified by a pipe line state or by a explicit delay

Creation of the central controller from the language ISAC 1/4

- Initialization
  - Let $\sum$ is an event
  - Let S be a set $S \subseteq \sum \times 2^{(\Sigma \times \mathbb{N})}$
  - Let *act* be a queue of events
    - insert event *main*
  - Let *cr* be a queue of operations in an instruction set
  - For every **Pipe Line** construction *i* create:
    - P_i: $St_i \rightarrow N_i$, where St_i is the set which contains names of states of the pipe line, $N_i = \{1,...,|St_i|\}$
- Other used notation in the algorithm

ev – an Event
st – State of a pipe line, in which is an operation accomplished
ACT – Activation
COD – Coding section of the operation
crev – Coding Root Event
GR – Goup
OP – Operation
CR – Coding Root

Creation of the central controller from the language ISAC 2/4

- Creating
  - while(*act* is not empty)
    - get *ev* from *act* and set *st* as the state of the pipe line
    - if (*ev* has **ACT**)
      - do *get_variants*(*ev*) $\subseteq 2^{(\Sigma \times \mathbb{N})}$
    - if (ev has **CR**)
      - insert events into *cr*
    - while(*cr* is not empty)
      - get *crev* from cr
      - do *get_variants_cr*(*crev*) $\subseteq 2^{(\Sigma \times \mathbb{N})}$
    - create relations in *S* between *ev* and values from *get_variants*, *get_variants_cr*

Creation of the central controller from the language ISAC 3/4

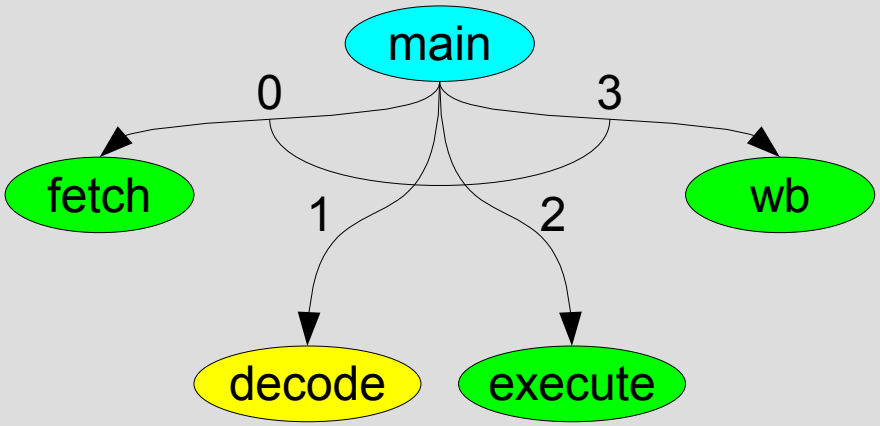
- Let *temp* be *temp* $\subseteq 2^{(\Sigma \times \mathbb{N})}$
- *get_variants*
  - for each event in **ACT**
    - for *ev* in **ACT** and compute delay and set *st* as the state of the pipe line
      - *delay* = *explicit* + *delay from pipe line*
      - create couple (*ev*, *delay*) and insert it into *temp*
      - insert *ev* into *act*
  - for each conditional activation in **ACT**
    - recursive do *get_variants* on all variants of conditional activation and do *combinate* results and *temp*
  - result is in *temp*

Creation of the central controller from the language ISAC 4/4

- Let *temp* be *temp* $\subseteq 2^{(\Sigma \times \mathbb{N})}$
- *get_variants_cr*
  - if (*crev* is **GR**)
    - recursive do *get_variants_cr* severally for all members in the group
    - returned values insert into *temp*
  - if (*crev* is **OP**)
    - each member in **COD** section insert into *cr*
    - if (*crev* has **ACT**)
      - do *get_variants*
      - *combinate* each member of *temp* which each member of the result of *get_variants*
  - result is in *temp*

Illustration 1/3

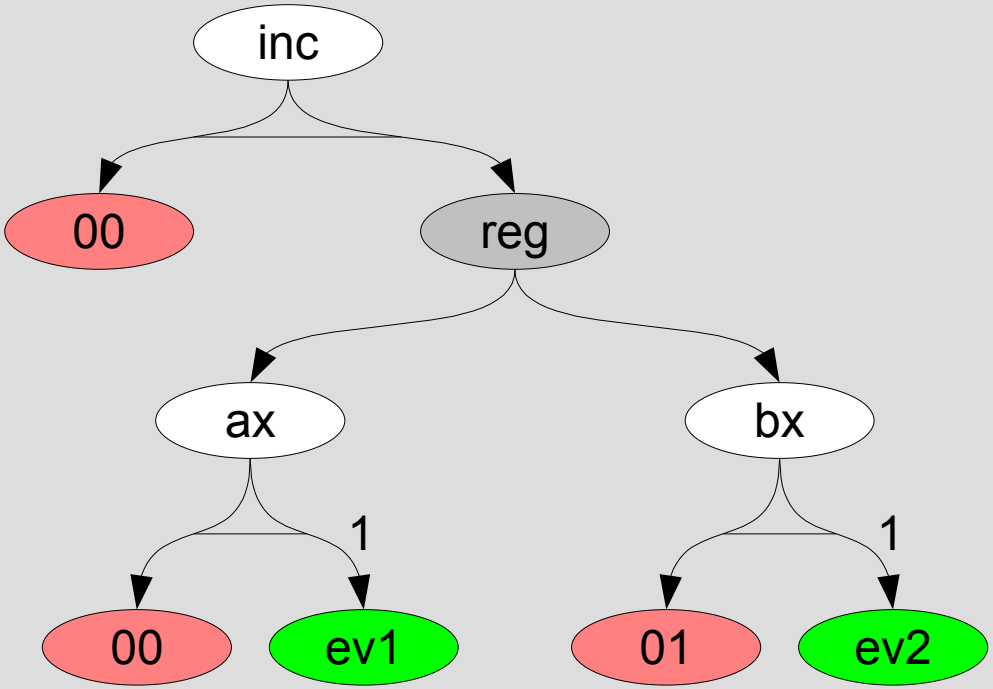
```
OP fetch {
    BEH {...}
}
OP decode {
    CR { inc(mem[pc]) }
}
OP execute {
    BEH {...}
}
OP wb {
    BEH {...}
}
OP main {
    ACT { fetch
          decode
          execute
          wb }
}
```



$S = \{(main, \{(fetch, 0), (decode, 1), (execute, 2), (wb, 4)\})\}$

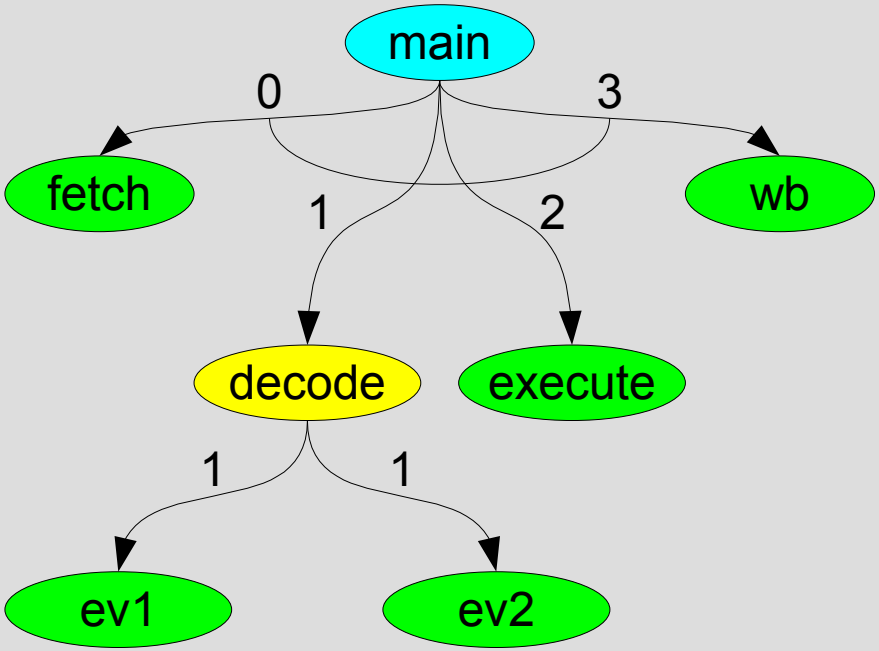
Illustration 2/3

```
OP ax {
    ASM { "ax" }
    COD { 0b00 }
    ACT { %1 ev1 }
}
OP bx {
    ASM { "bx" }
    COD { 0b01 }
    ACT { %1 ev2 }
}
GR reg = ax, bx
OP inc {
    ASM { "inc" reg }
    COD { 0b01 reg }
}
```



temp = {{(ev1, 1)}, {(ev2, 1)}}

Illustration 3/3



S = {(*main*, {(*fetch*, 0), (*decode*, 1), (*execute*, 2), (*wb*, 4)}), (*decode*,{(*ev1*, 1)}), (*decode*, {(*ev2*, 1)})}

Conclusion

- Fast and simply algorithm
- No dynamic planning
- Easy transformation into language C or HDL
- Central controller is used for simulation
- Central controller can be synthesized into HW