PIPELINE DESIGN

DATA PATH AND CONTROL SYNTHESIS

©*Giovanni De Micheli*

Stanford University

Outline

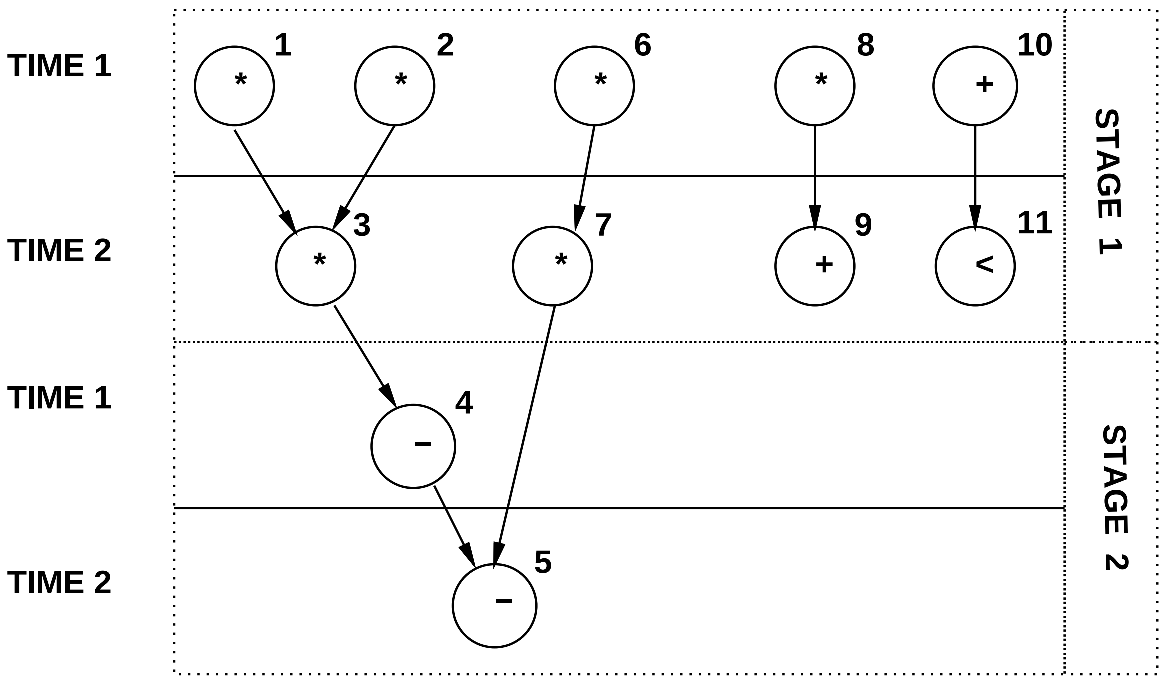
- Synthesis of pipelined circuits:
  - Scheduling.
  - Binding.
- Data-path synthesis.
- Control-unit synthesis.

High-level synthesis of pipelined circuits

- Pipeline circuits:
  - Concurrent execution of operations on different data sets.
  - Increase *throughput*:
    - I/O data rate.
  - Preserve *latency*.

- Applicable to:
  - General purpose processors.
  - Digital signal processors.

Example



Synthesis of pipelined circuits

- DSP applications:
  - Mainly data-path pipelining.
  - Few exceptions/interrupts.
  - Mature area.
- Microprocessors:
  - Advanced features:
    - *Stalls, flush, bypass, hazard* avoidance.
  - Synthesis tools not ready yet.

Issues in synthesis of pipelined circuits

- Partitioning:
  - *Pipe-stage* formation.
- Scheduling:
  - Source vertex of the sequencing graph fired at constant rate.
- Sharing:
  - More concurrency.
  - Binding and scheduling are affected.

Scheduling of pipelined circuits

© GDM

- Scheduling of non-pipelined circuit using pipelined resources.

- Scheduling of pipelined circuit using non-pipelined resources.

  – Functional pipelining.

- Both problems can be modeled by ILP.

Scheduling for functional pipelining

- Choose:
  - *cycle-time*.
  - *data-introduction interval* δ_0.
- Determine *(area,latency)* spectrum.
- Key fact:
  - Simultaneous operations at steps:
    - $l + p\delta_0$
  - Reduced sharing.

Scheduling for functional pipelining ILP model

$$\sum_{p=0}^{\lceil \overline{\lambda}/\delta_0 \rceil - 1} \sum_{i:\mathcal{T}(v_i)=k} \sum_{m=l-d_i+1+p\delta_0}^{l+p\delta_0} x_{im} \leq a_k \quad \forall k, \forall l$$

- Used in conjunction with other constraints.

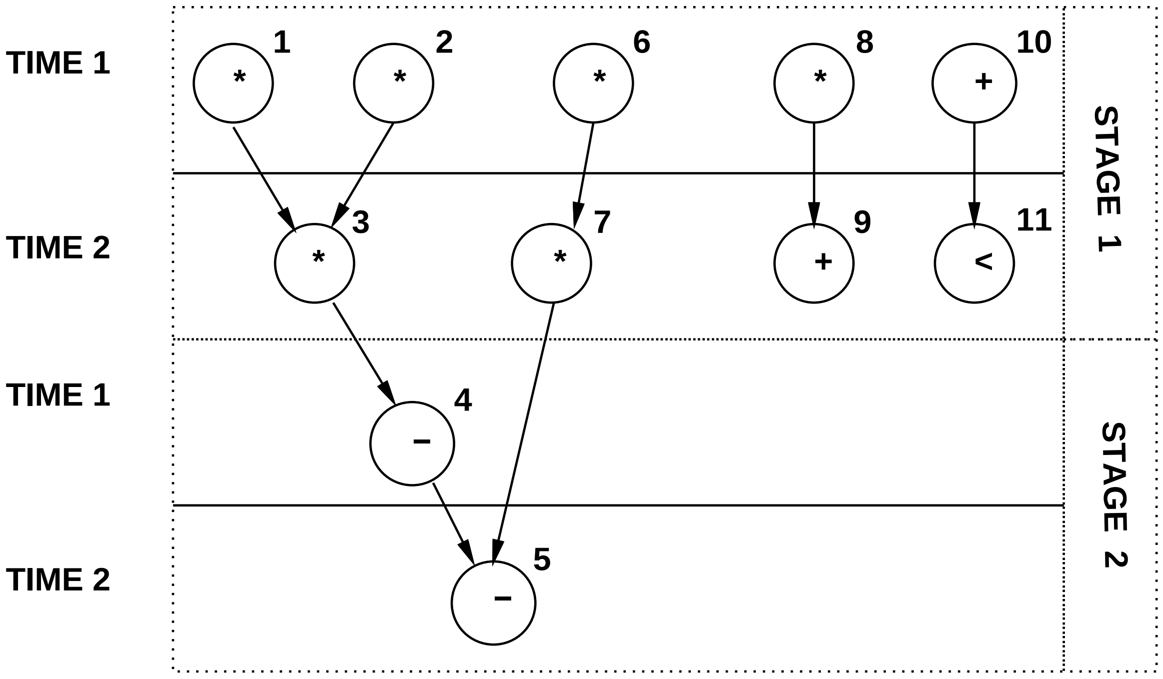
- Use regular ILP solvers.

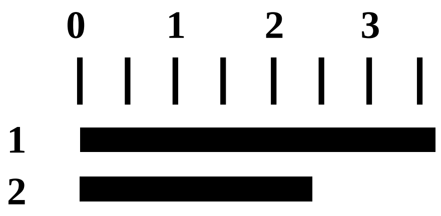
Scheduling for functional pipelining Heuristic algorithms

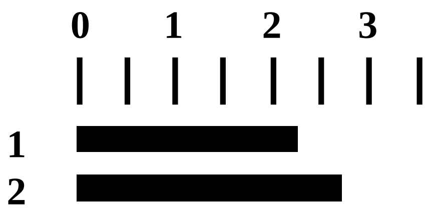
- List scheduling:
  - Compute resource usage at each step.
  - Determine candidates.
- Force-directed scheduling.
  - Operation-type distribution:
    - ∗ Account for overlapping.

Example

© GDM





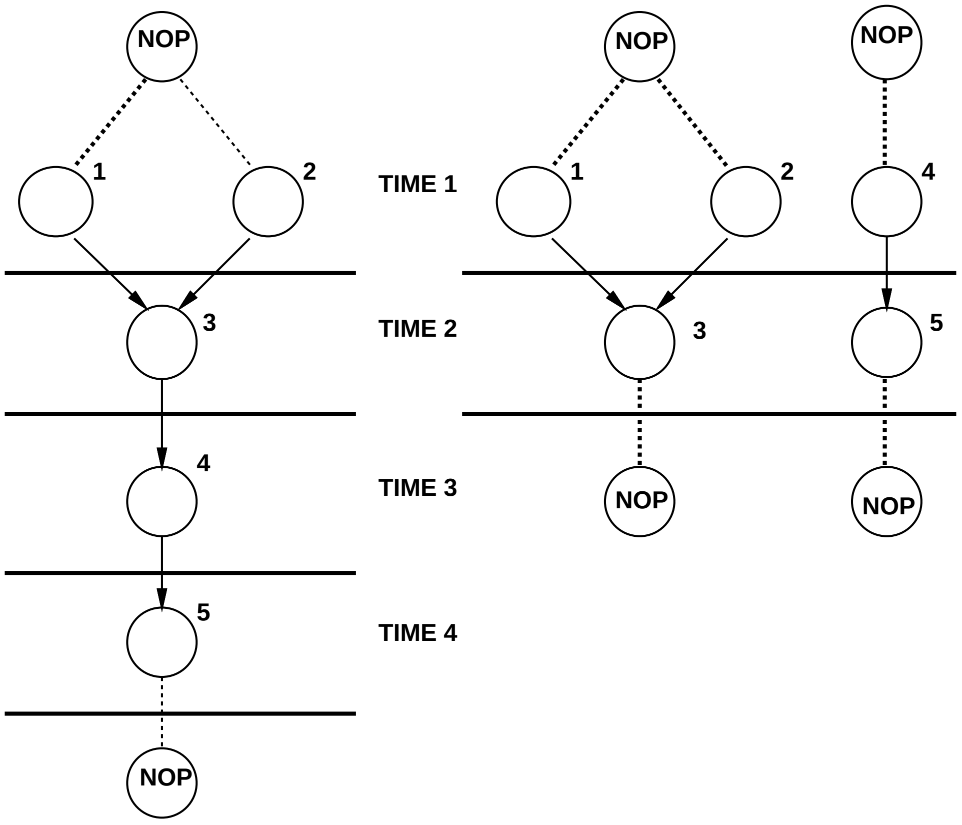


- Distribution graphs for multiplier and ALU.

Loop folding

- Reduce execution delay of a loop.
- Pipeline operations inside a loop.
  - Overlap execution of operations.
  - Need a prologue and epilogue.
- Use pipeline scheduling for loop graph model.

Example

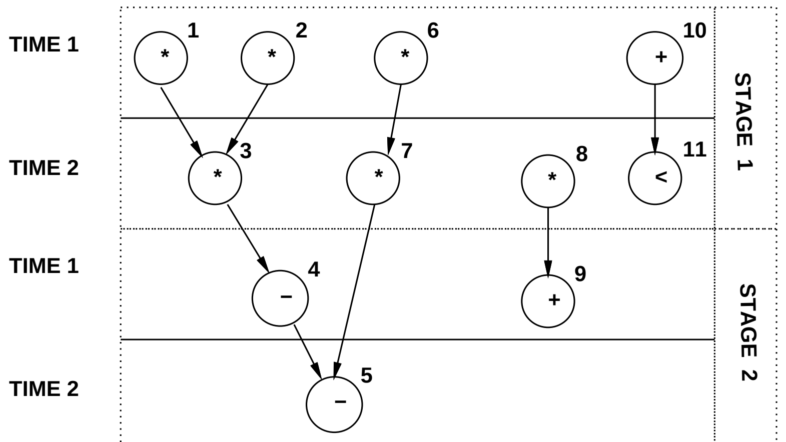


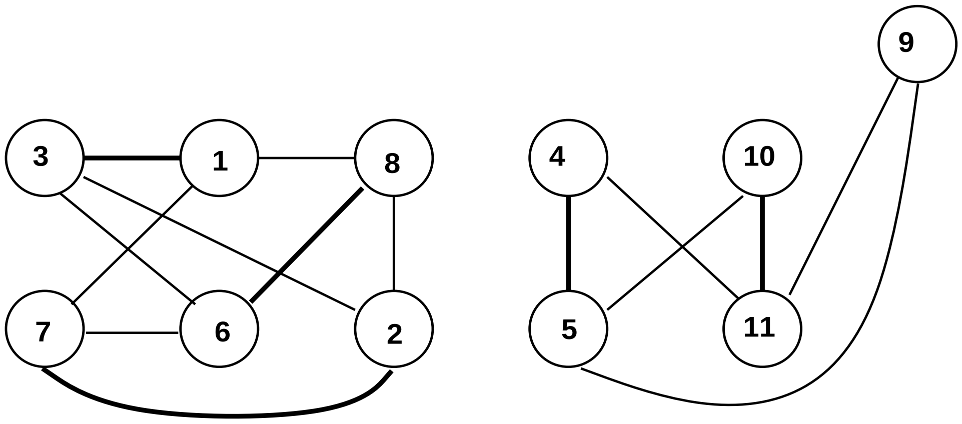
(a) (b)

Resource sharing for pipelined circuits

- Scheduled graphs:
  - Determine compatibility (or conflict) graphs.
- The lower the δ_0 (the higher the *throughput*):
  - The lower the compatibility.

Example: $\delta_0 = 2$

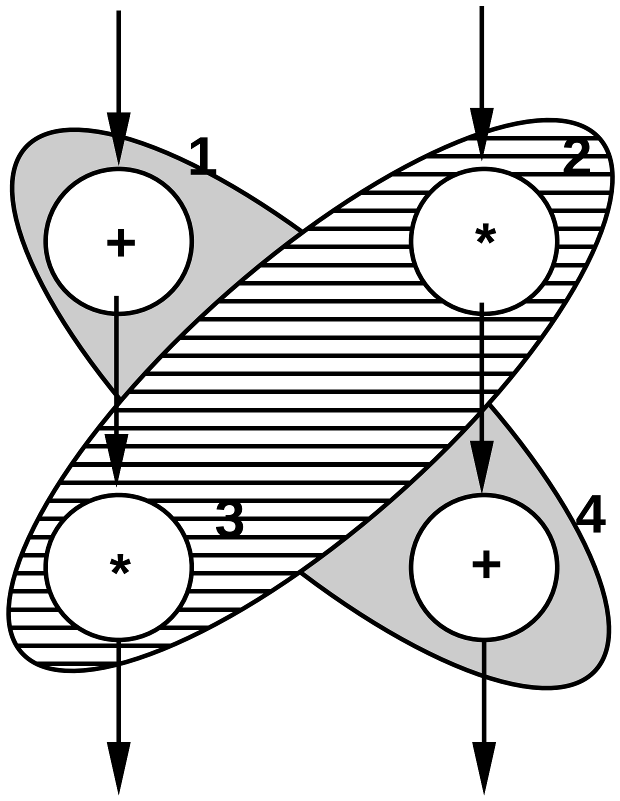




Resource sharing for pipelined circuits

- Branching constructs:
  - Special care to avoid *deadlocks*.
- *Twisted pairs*:
  - Two mutually compatible operation pairs with twisted dependencies.
- Sharing operations in twisted pairs must be avoided.

Example



Data path synthesis

- Resource binding.

- Connectivity synthesis:

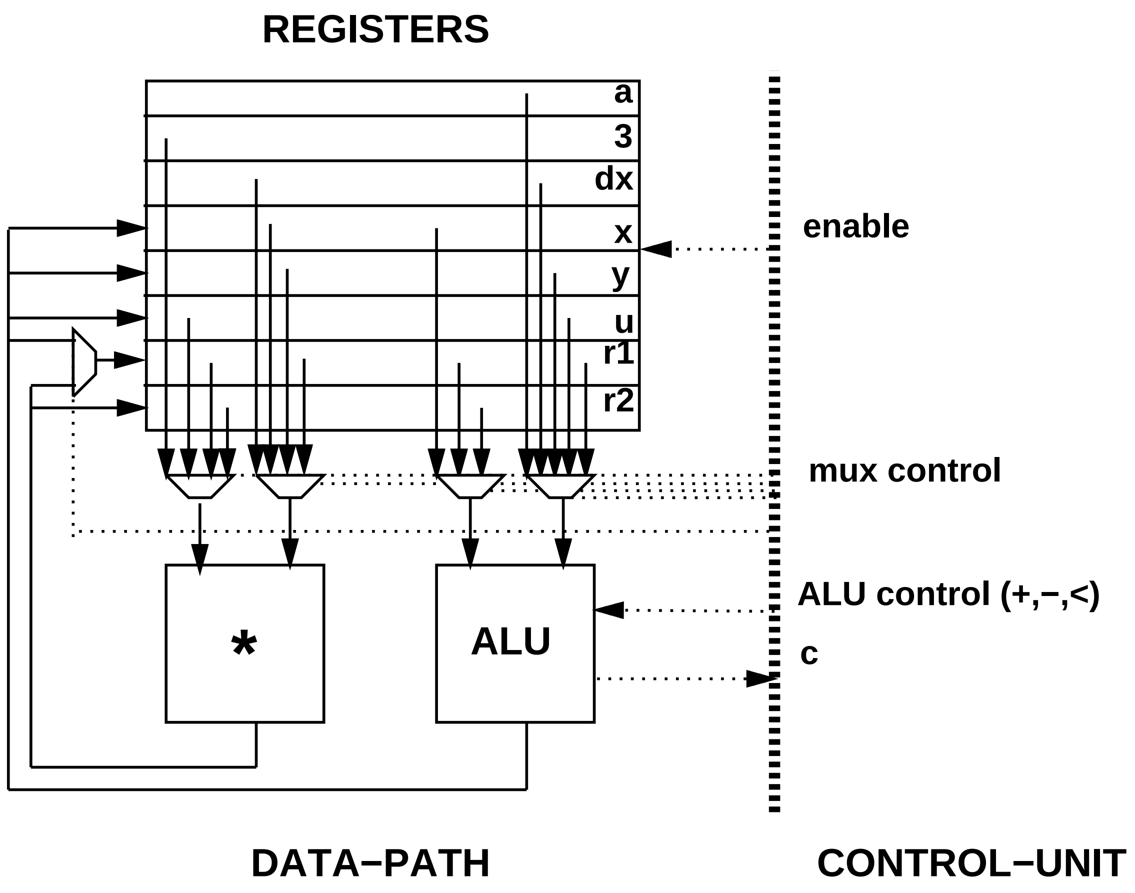
  - Connection of resources to: *multiplexers busses* and *registers.*

  - Control unit interface.

  - I/O ports.

- Physical data-path synthesis.

Example


REGISTERS
a
3
dx
x
y
u
r1
r2
enable
mux control
ALU control (+,−,<)
c
*
ALU
DATA−PATH
CONTROL−UNIT

Control synthesis

- Synthesis of the control unit.
- Logic model:
  - Synchronous FSM.
- Physical implementation:
  - Microcode (ROM,PLA).
  - Hard-wired FSM.
  - Distributed FSM.

Control synthesis

- Synthesize circuit that:
  - Executes scheduled operations.
  - Provides synchronization.
  - Supports:
    - Iteration.
    - Branching.
    - Hierarchy.
    - Interfaces.
- Assumption:
  - Synchronous implementation.
  - Control unit is a FSM (or connection of FSMs).

Controlling scheduled operations

- Simple model:
  - No branching, iteration, hierarchy.
  - No data-dependent delays.
- Implementation:
  - FSM-oriented design:
    - Hardware: PLAs, gates, registers.
    - One FSM state per schedule level.
  - Microcode-oriented design:
    - Hardware: ROM, PLA, counter.

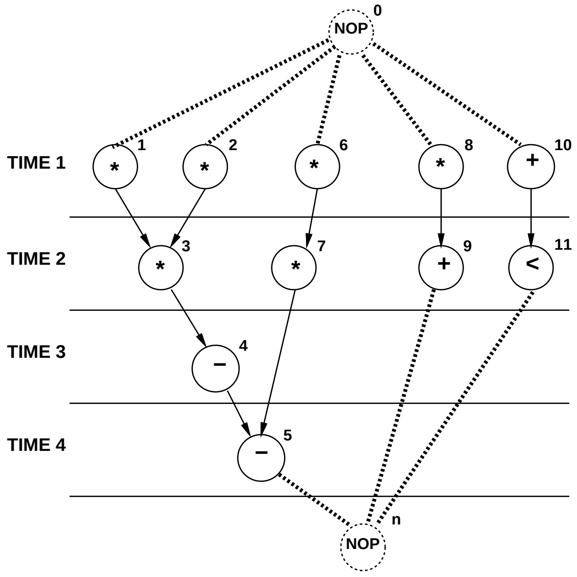
FSM-based implementation

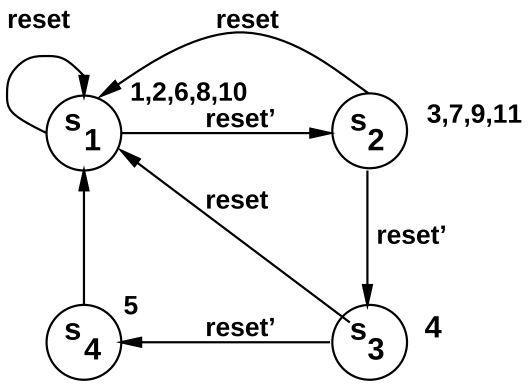
- Simple model:
  - *next-state function*: unconditional.
  - *output function*: activate operations.

- Extended model:
  - Branching and iteration:
    - Conditional next-state function.
  - Hierarchy:
    - Hierarchical FSM connection.

Example

© GDM

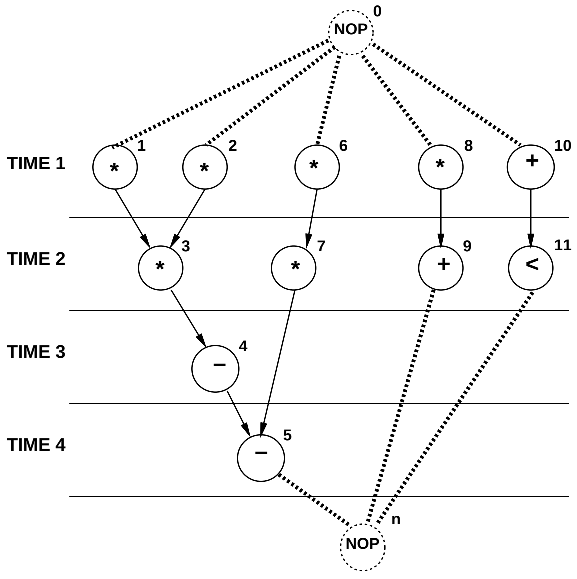

NOP
0
TIME 1
TIME 2
TIME 3
TIME 4
*
1
*
2
*
6
*
8
+
10
*
3
*
7
+
9
<
11
−
4
−
5
NOP
n

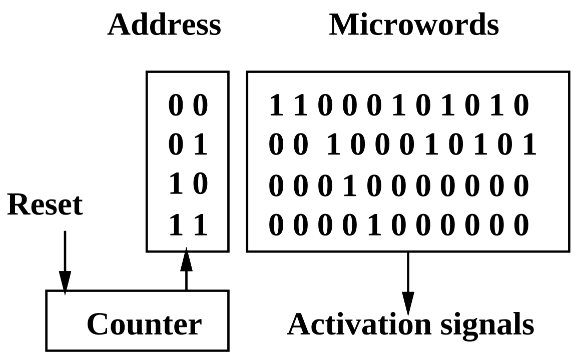

reset
reset
1,2,6,8,10
reset'
s_1
s_2
3,7,9,11
reset
reset'
5
s_4
reset'
s_3
4

Microcode implementation

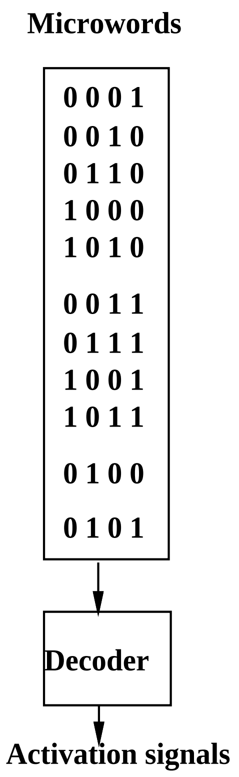
- Horizontal microcode:
  - One bit per *activation* signal.
  - One microcode word per schedule level.
  - Maximum performance.
  - Wide words.
- Vertical microcode:
  - Encode each resource *activation* signal.
  - Shorter words.
  - One (or more) words per schedule level.

Example of horizontal microcode





Example of vertical microcode



Microcode compaction problem

- Partition ROM word into fields.
- Encode signals in each field.
- Allow for a code for NOP.
- Activation signals in each field must not be
  concurrent.
- Problems:
  - Minimize number of fields.
  - Minimize total ROM width.

Microcode optimization

- Conflict graph:
  - Concurrent operations.
  - Optimum *vertex coloring* yields minimum number of *fields*.
- Compatibility graph:
  - Non-concurrent operations.
  - Optimum *clique partitioning* yields minimum number of *fields*.
  - Minimum *weighted* clique partitioning yields minimum number of *bits*.

Example

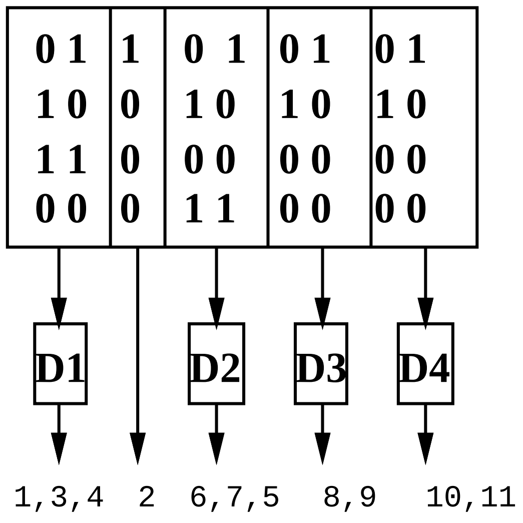
field	op	code
A	1	01
A	3	10
A	4	11
B	2	1
C	6	01
C	7	10
C	5	11
D	8	01
D	9	10
E	10	01
E	11	10

Example

Microword format

A	B	C	D	E

Microwords



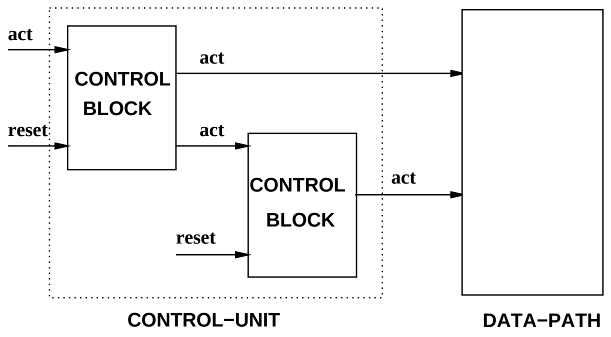
Activation signals

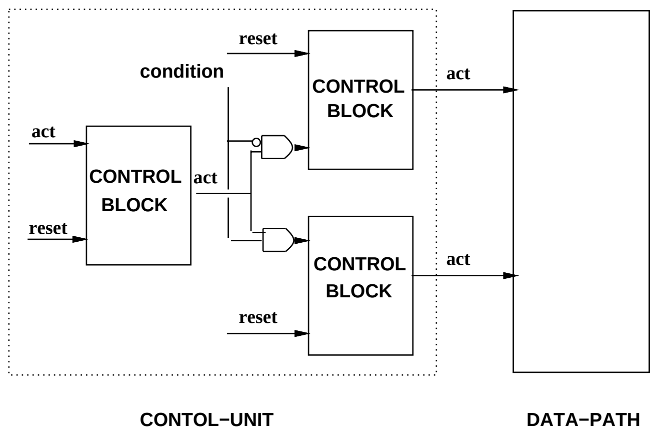
Hierarchical control

- Exploit the hierarchical structure of sequencing graphs.
- One controller per entity.
- Interconnected *finite state machines*.
- Handshake:
  - *activate* signals.
  - *condition* signals.
  - *reset* signals.

Example

© GDM


act
reset
CONTROL BLOCK
act
act
CONTROL BLOCK
reset
act
CONTROL−UNIT
DATA−PATH


reset
condition
CONTROL BLOCK
act
act
reset
CONTROL BLOCK
act
CONTROL BLOCK
act
reset
CONTOL−UNIT
DATA−PATH

Control synthesis for unbounded-latency sequencing graphs

- Data-dependent delay operations.
  - *activate signals*.
  - *completion signals*.
- Synchronization problem:
  - Wait on *completion signals*.
  - Wait on external synchronization.
- Several strategies.
  - Clustering.
  - Adaptive Control.
  - Relative Scheduling.

Summary
Control synthesis

- Different approaches.

- Implementations:
  - FSM, connection of FSMs or ROM.

- Techniques:
  - Bounded delays only:
    - FSM – microcode.
  - Unbounded delays:
    - Different methods to provide synchronizatio