Mod 6.

goals.

- ☐ Perform a Control & Data Analysis on a C program
- ☐ Analyze Behavior of control flow graphs CFG's and Data flow Graphs DFG's
- ☐ Map C program constructs to CFG's & DFG's.
- ☐ Compose & Modify a VHDL description

} Chap 4 of text.

S1: C cannot be directly mapped to parallel

Understand C operation in

context to of: Data Edge (data out by 1 = data in by 2)

Control Edge (order of operation)

S2:

```
int max(int a,b)
{
int r;
if (a>b)
    r=a;
else
    r=b;
return r;
}
```

op1 — entrance to function

op2 — if, then, else

op3

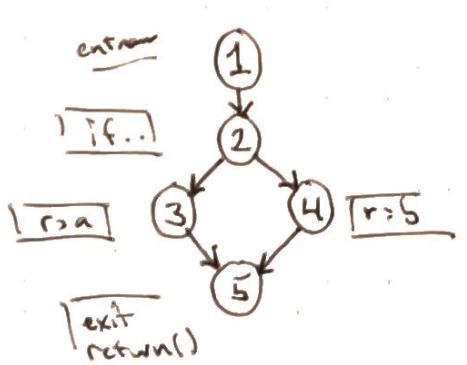
op4

op5 — return / exit

operations

— find 'control edges'

find all possible paths through program...



CFG

Data Flow Graph

[53]

```
int max(int a,b) {
    int r;
    if (a>b)
        r=a;
    else
        r=b;
    return r;
}
```

op 1 = produces a, b

op 2 = consumes a, b

op 3 = consumes a, and (a>b),
produces r

op 4 = consumes b, and (a>b),
produces r

op 5 = consumes r

DFG: production & consumption of patterns for each variable

i.e variable a is read by ops 2 & 3	variable b is read by ops 2 & 4

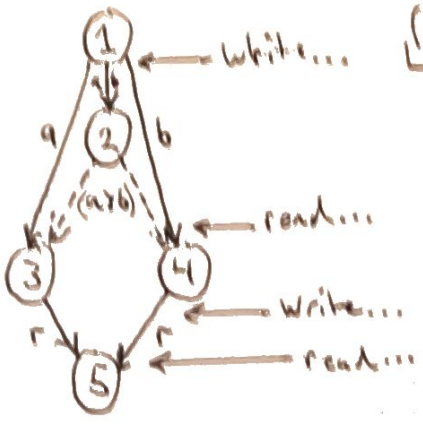
Produces 'data' edges from 1→2 (a & b), 1→3 (a), 1→4 (b)

DFG extends CFG

[54]

[55]

Data flow Graph for above



DFG's 'edge's or 'arc's are the variables

[S6] CFG's & DFG's

Capture the behavior of a 'c' program 'graphically'.

How can we go from CFG / DFG to Hardware.

- Data edges reflect requirements of flow...

- Control edges, provide mechanism to distill algorithm to sequence of operations

As humans, we are wired to think sequentially, we have to change how we think for parallel hardware.

Taking CFG's & DFG's beyond...
we can leave behind order of operation, Control edges therefore will be left behind. Data edges must be kept.

f.n. keep in mind a CFG/DFG for k-means ~~nearest~~ cluster.

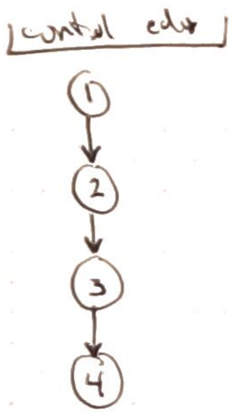
[S7]

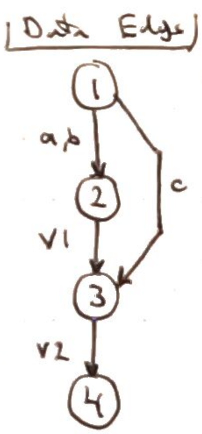
Implementation

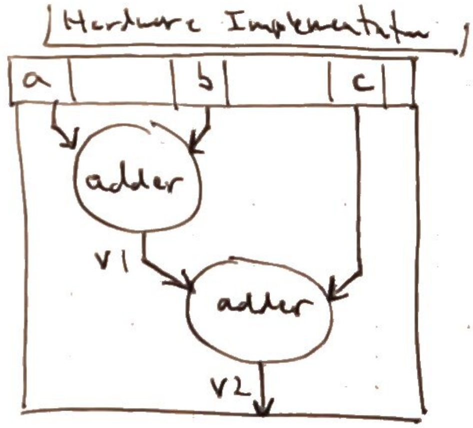
parallel architecture can be executed out of order...
BUT, MUST always preserve data dependencies.

~ C progr. ~

```
int sum(int a,b,c){
  int v1;
  v1 = a+b;
  v2 = v1+c;
  return v2; }
```







above all, when mapping 'c', must preserve data dependencies

Sequential order from CFG is eliminated.

Both additions in single clock cycle.

C to CFG

WORKFLOW

let:

- node, ◎, represent operations ('c' statements)
- edges →, ←, ↔, represent execution order between two connected nodes (operations)

C executes sequentially, so this is usually straight forward

Exception occurs when multiple control edges originate from single operation

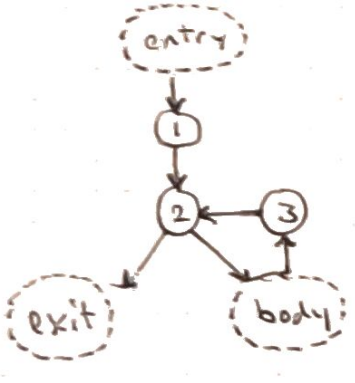
C code:

```
for (i = 0; i < 20 ; i ++) {
    // body of loop
}
```

Above statement..., 4 parts

- loop initialization
- loop condition
- loop counter increment operation
- body of loop (internals)

59 translate above 'c' to control flow graph




```
    ①   ②   ③
for (i=0; i<20; i++) {
    // ~~
}
```

dashed == single entry, single exit.

① = initialization

② = conditions check

③ = increment

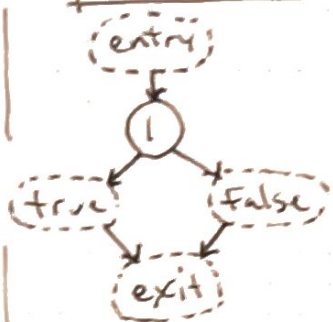
S9.

do-while and while-do

A]

```
①
if (a < b) {
   // True body
}
else {
   // False branch
}
```

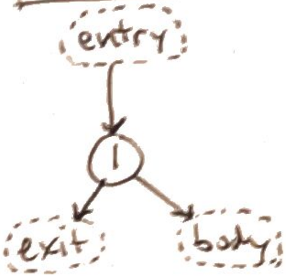
if, then, else



B]

```
①
while (a < b) {
    // loop body
}
```

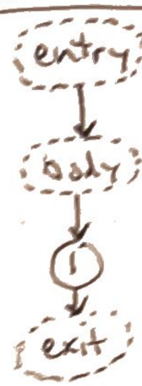
While loop **



C]

```
do {
  // loop body
} while (a < b)
             ①
```

do while



F.N.
Song "Just dropped in"
Kenny Rogers "what condition my condition was in"

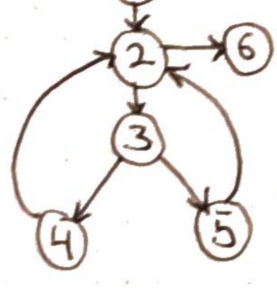
* do-while always executes body at least once

** while only executes body if condition is met.

S10 Greatest Common Divisor GCD

Control flow Graph ...

```
1 | int gcd(int a, int b) {
2 |     while (a != b) {        ***
3 |         if (a > b)
4 |             a = a - b;
  |         else
5 |             b = b - a;
  |     }
6 | return a; }
```



Euclid's Algorithm

*** will only execute body if a != b and stops when a != b false. otherwise exit.

each non terminating →

Each non terminating path
will follow one of these two...

either 2 >> 3 >> 4 >> 2

or

2 >> 3 >> 5 >> 2

control paths help construct DFG

S11

C program to Data flow Graph

Same # nodes as CFG.
edges will be different.

Possible to go from C → DFG
but better to go C → CFG → DFG

Trace control paths in CFG
while simultaneously identify read/write operations of variables

focussing on C w/o arrays
w/o pointers.

CFG does NOT label edges w/ variables

DFG does label edges w/ variables.

S12

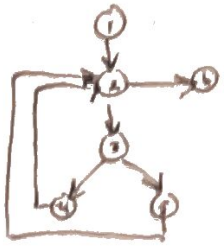
usable Workflow.

1) start where a variable is read
2) identify CFG nodes that assign to that variable (write nodes)
3) Introduce a 'Data Edge' between a read and a write node under the condition that the control path does NOT pass through another write-node for that variable.
4) repeat for all read-nodes

process identifies edges related to assignment statements BUT NOT those originating from conditional expressions in control flow statements.

DFG for GCD

① Nodes from CFG =



Data Edges discovered will be either result of assignment statements or conditional expressions

```
1) int gcd(int a, int b) {
2)   while (a != b) {
3)     if (a > b)
4)       a = a - b;
       else
5)       b = b - a;
     }
6)   return a;
   }
```

② Pick a node where a variable is read.

i.e. Node 5.

```
b = b - a
```

b is read

a is read

b is also written back to ...

arbitrarily chose b.

Now, trace backwards through predecessors of 5 in CFG, that return b

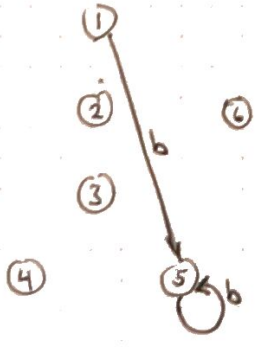
~~this gives 5~~ 3, 2, 1, 4, 5 (not 6)

Nodes 1, 5 write b

there is a direct path, lexically from 1 to 5.

and a direct path from 5 to 5 (5, 2, 3, 5)

So: add data edges from 1, 5 & 5, 5



'⟶' = assignment only data edge

'⇢' = conditional data edge

Cont.

Now Variable 'a' also starting

@ Node 5

Trace backwards till we find a 'write' to 'a'

4 & 1 write to 'a'

4 also reads 'a'

3 reads 'a'

2 reads 'a'

1 writes 'a'

5 reads 'a'

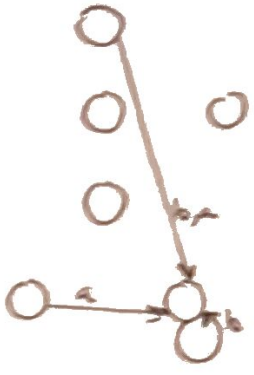
6 reads 'a'

Only Concerned arrows w/ node 5.

trace back to the nodes that write 'a'

SO 1 → 5

& 4 → 5

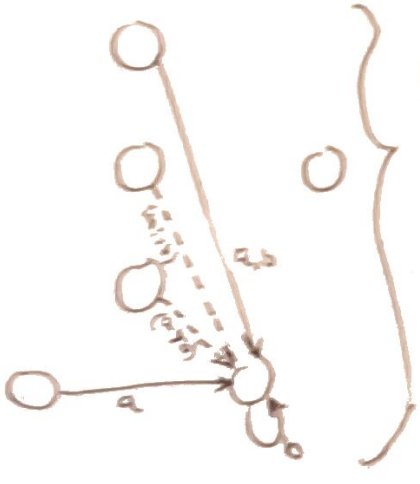


Node 5 'b' assignments done ☑ } Solid

Node 5 'a' assignments done ☑ }

Node 5 (N2) conditional ___ ☐ } dotted

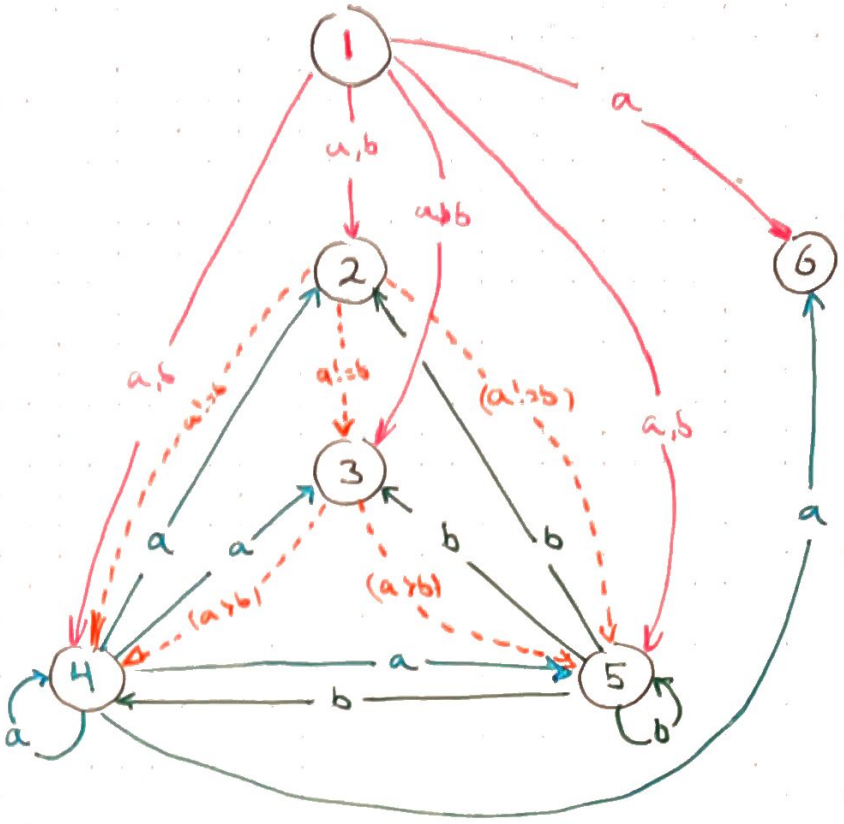
Node 5 (N2) conditional ___ ☐ }



Node 5
complete
all ~~the~~ variable assignments
all conditional dependencies

repeat
for all
other nodes.

Partial representation of all nodes..



Successfully went from 'C'
to CFG
to DFG

from DFG, can now go to hardware...

Book. 4.5

Application: Translating C to Hardware

Recall, starting w/ C, make CFG, then DFG.

With 'C', CFG, & DFG in Hand...

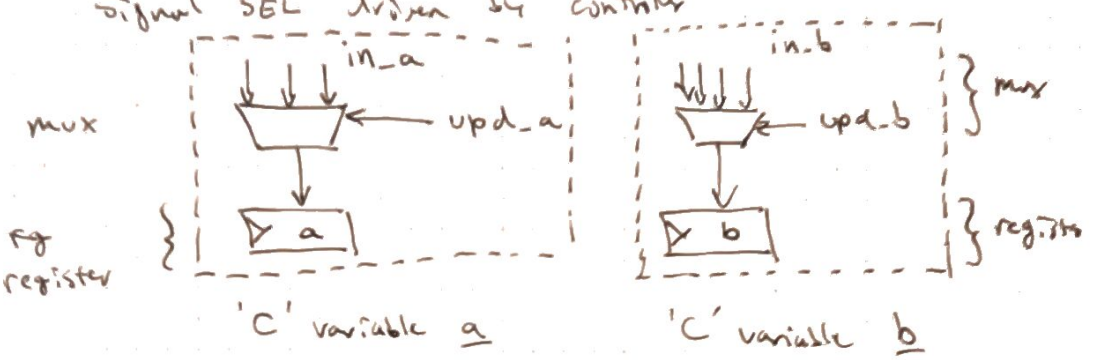
1) make each variable in C a

register with multiplexer in front

multiplexers necessary when multiple sources can input to variable.

Also, the register will update itself.

signal SEL driven by controller



upd_a, upd_b } driven by controller

2) For each 'C' expression INSIDE A NODE of CFG

do

make an equivalent combinational circuit to implement that expression.

i.e. $b = b - a$

combinational logic = subtractor

if conditionals like greater than, less than, etc. these also produce additional data path elements in the form of boolean flags. These flags are used by the controller of the data path.