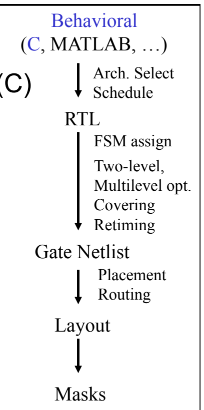
ESE535: Electronic Design Automation

Day 14: March 14, 2011
C→RTL

Today

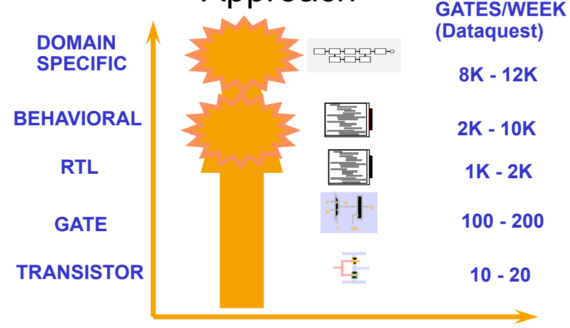


See how get from a language (C) to dataflow

- Straight-line code
- If-conversion
- Memory
- Basic Blocks
- Control Flow
- Looping
- Hyperblocks
- Common Optimizations

Day 1

Design Productivity by Approach



Source: Keutzer (UCB EE 244)

Arithmetic Operators

• Unary Minus (Negation)	-a
• Addition (Sum)	a + b
• Subtraction (Difference)	a - b
• Multiplication (Product)	a * b
• Division (Quotient)	a / b
• Modulus (Remainder)	a % b

Things might have an a hardware operator for…

Bitwise Operators

• Bitwise Left Shift	a << b
• Bitwise Right Shift	a >> b
• Bitwise One's Complement	~a
• Bitwise AND	a & b
• Bitwise OR	a | b
• Bitwise XOR	a ^ b

Things might have an a hardware operator for…

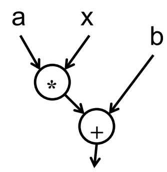
Comparison Operators

• Less Than	a < b
• Less Than or Equal To	a <= b
• Greater Than	a > b
• Greater Than or Equal To	a >= b
• Not Equal To	a != b
• Equal To	a == b
• Logical Negation	!a
• Logical AND	a && b
• Logical OR	a || b

Things might have an a hardware operator for…

Expressions: combine operators

- a*x+b



A connected set of operators
→ Graph of operators

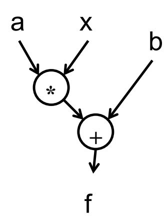
Expressions: combine operators

- a*x+b
- a*x*x+b*x+c
- a*(x+b)*x+c
- ((a+10)*b < 100)

A connected set of operators
→ Graph of operators

C Assignment

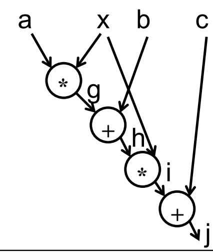
- Basic assignment statement is:
  Location = expression
- f=a*x+b



Straight-line code

- a sequence of assignments
- What does this mean?

g=a*x;
h=b+g;
i=h*x;
j=i+c;



Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable

t=a*x;
r=t*x;
t=b*x;
r=r+t;
r=r+c;

Variable Reuse

- Variables (locations) define flow between computations
- Locations (variables) are reusable

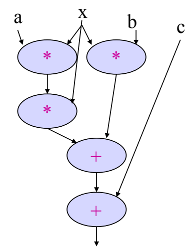
t=a*x;	t=a*x;
r=t*x;	r=t*x;
t=b*x;	t=b*x;
r=r+t;	r=r+t;
r=r+c;	r=r+c;

- Sequential assignment semantics tell us which definition goes with which use.
  - **Use** gets most recent preceding **definition**.

Dataflow

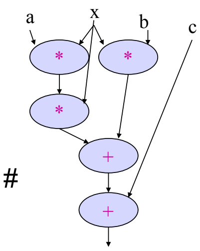
- Can turn sequential assignments into dataflow graph through def→use connections

t=a*x;   t=a*x;
r=t*x;   r=t*x;
t=b*x;   t=b*x;
r=r+t;   r=r+t;
r=r+c;   r=r+c;



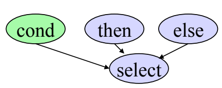
Dataflow Height

- t=a*x;   t=a*x;
  r=t*x;   r=t*x;
  t=b*x;   t=b*x;
  r=r+t;   r=r+t;
  r=r+c;   r=r+c;
- Height (delay) of DF graph may be less than # sequential instructions.



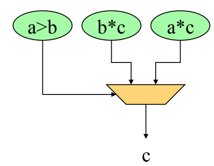
Simple Control Flow

- If (cond) { … } else { …}
- Assignments become conditional
- In simplest cases, can treat as dataflow node



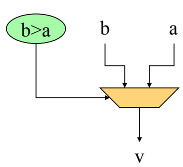
Simple Conditionals

if (a>b)
  c=b*c;
else
  c=a*c;



Simple Conditionals

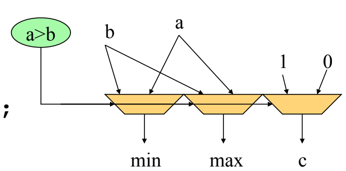
v=a;
if (b>a)
  v=b;



- If not assigned, value flows from before assignment

Simple Conditionals

```
max=a;
min=a;
if (a>b)
  {min=b;
   c=1;}
else
  {max=b;
   c=0;}
```



- May (re)define many values on each branch.

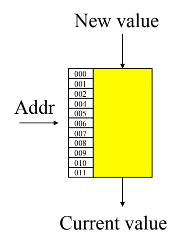
Lecture Checkpoint

- Happy with
  - Straight-line code
  - Variables
  - Conditionals

- Next topic: Memory

C Memory Model

- One big linear address space of locations
- Most recent definition to location is value
- Sequential flow of statements



C Memory Operations

Read/Use
- a=*p;
- a=p[0]
- a=p[c*10+d]

Write/Def
- *p=2*a+b;
- p[0]=23;
- p[c*10+d]=a*x+b;

Memory Operation Challenge

- Memory is just a set of location
- But **memory expressions** can refer to variable locations
  - Does *q and *p refer to same location?
  - *p and q[c*10+d]?
  - p[0] and p[c*10+d]?
  - p[f(a)] and p[g(b)] ?

Pitfall

- P[i]=23
- r=10+P[i]
- P[j]=17
- s=P[j]*12

- Value of r and s?

- Could do:
  P[i]=23; P[j]=17;
  r=10+P[i]; s=P[j]*12

….unless i==j
Value of r and s?

C Pointer Pitfalls

- *p=23
- r=10+*p;
- *q=17
- s=*q*12;

- Similar limit if p==q

C Memory/Pointer Sequentialization

- Must preserve ordering of memory operations
  - A read cannot be moved before write to memory which may redefine the location of the read
    - Conservative: any write to memory
    - Sophisticated analysis may allow us to prove independence of read and write
  - Writes which may redefine the same location cannot be reordered

Consequence

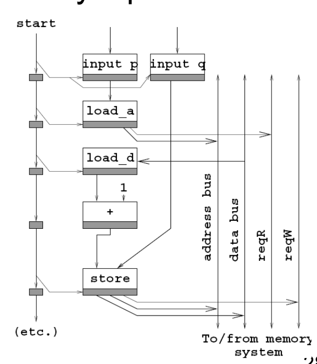
- **Expressions and operations** through variables (whose address is never taken) can be executed at any time
  - Just preserve the dataflow
- **Memory assignments** must execute in strict order
  - Ideally: partial order
  - Conservatively: strict sequential order of C

Forcing Sequencing

- Demands we introduce some discipline for deciding when operations occur
  - Could be a FSM
  - Could be an explicit dataflow token
  - Callahan uses control register
- Other uses for timing control
  - Variable delay blocks
  - Looping
  - Complex control

Scheduled Memory Operations

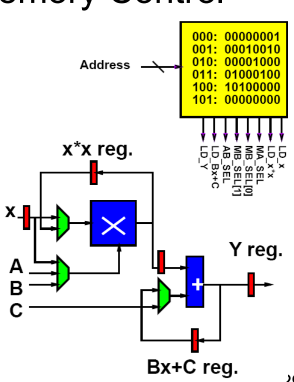
```
*q = *p + 1;
(etc.)
```



Source: Callahan

Day 3 Quadratic Memory Control

1. LD_X
2. MA_SEL=x, MB_SEL[1:0]=x, LD_x*x
3. MA_SEL=x, MB_SEL[1:0]=B
4. AB_SEL=C, MA_SEL=x*x, MB_SEL=A, LD_Bx+C
5. AB_SEL=Bx+C, LD_Y



Basic Blocks

- Sequence of operations with
  - Single entry point
  - Once enter execute all operations in block
  - Set of exits at end

```
A=B+C
E=A*D
If (E>100)
  {
     Q++;
     E=E-100;
  }
G=F*E;
```

```
BB0:
  A=B+C
  E=A*D
  t=(E>100)
  br(t,BB1,BB2)
```

```
BB1:
  Q++
  E=E-100
  br BB2

BB2:
  G=F*E
```

Basic Blocks?

Basic Blocks

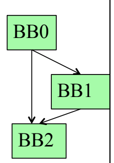
- Sequence of operations with
  - Single entry point
  - Once enter execute all operations in block
  - Set of exits at end
- Can dataflow schedule operations within a basic block
  - As long as preserve memory ordering

Connecting Basic Blocks

- Connect up basic blocks by routing control flow token
  - May enter from several places
  - May leave to one of several places

Connecting Basic Blocks

- Connect up basic blocks by routing control flow token
  - May enter from several places
  - May leave to one of several places



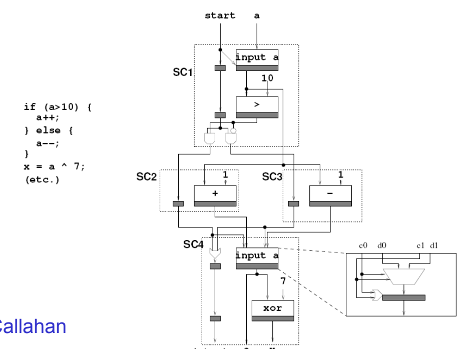

```
A=B+C
E=A*D
If (E>100)
  {
    Q++;
    E=E-100;
  }
G=F*E;
```

```
BB0:
  A=B+C
  E=A*D
  t=(E>100)
  br(t,BB1,BB2)
```

```
BB1:
  Q++
  E=E-100
  br BB2

BB2:
  G=F*E
```

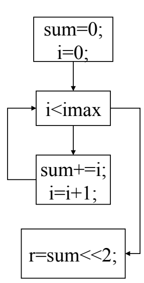

Basic Blocks for if/then/else



Source: Callahan

Loops

```
sum=0;
for (i=0;i<imax;i++)
    sum+=i;
r=sum<<2;
```

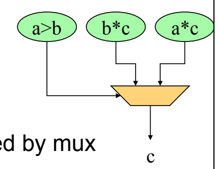


Beyond Basic Blocks

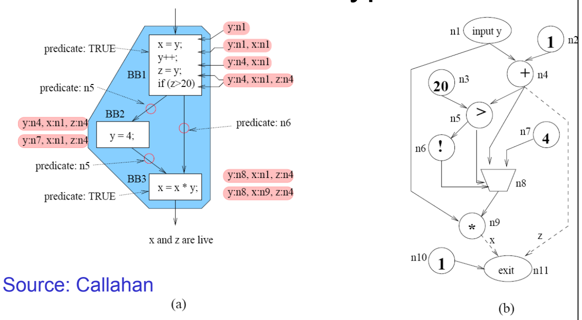
- Basic blocks tend to be limiting
- Runs of straight-line code are not long
- For good hardware implementation
  - Want more parallelism

Hyperblocks

- Can convert if/then/else into dataflow
  - If/mux-conversion
- Hyperblock
  - Single entry point
  - No internal branches
  - Internal control flow provided by mux conversion
  - May exit at multiple points



Basic Blocks → Hyperblock

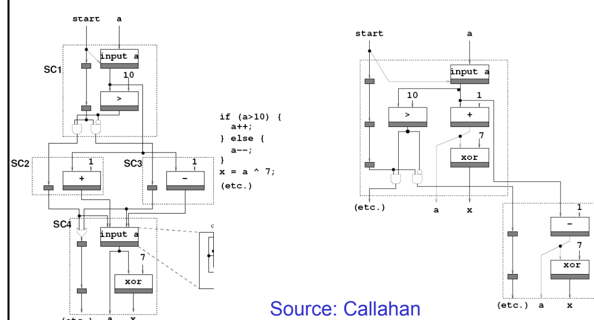


Source: Callahan

Hyperblock Benefits

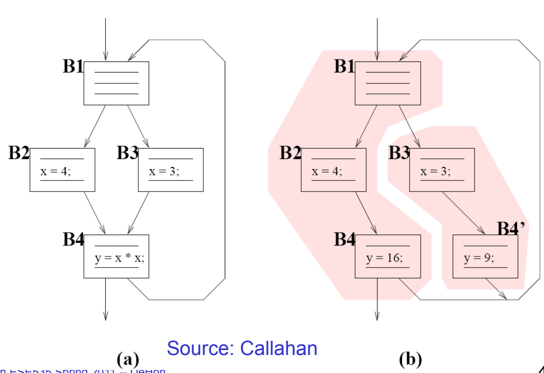
- More code → typically more parallelism
  - Shorter critical path
- Optimization opportunities
  - Reduce work in common flow path
  - Move logic for uncommon case out of path
    - Makes smaller faster

Common Case Height Reduction



Source: Callahan

Common-Case Flow Optimization

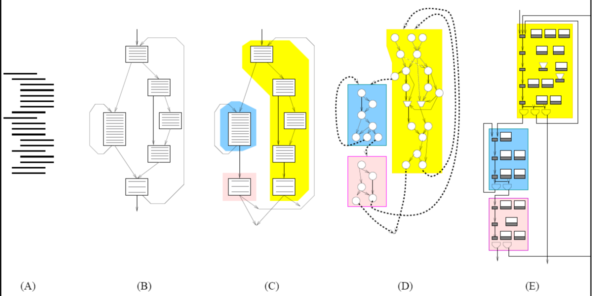


Source: Callahan

Optimizations

- Constant propagation: a=10; b=c[a];
- Copy propagation: a=b; c=a+d; → c=b+d;
- Constant folding: c[10*10+4]; → c[104];
- Identity Simplification: c=1*a+0; → c=a;
- Strength Reduction: c=b*2; → c=b<<1;
- Dead code elimination
- Common Subexpression Elimination:
  - C[x*100+y]=A[x*100+y]+B[x*100+y]
  - t=x*100+y; C[t]=A[t]+B[t];
- Operator sizing: for (i=0; i<100; i++) b[i]=(a&0xff+i);

Flow Review



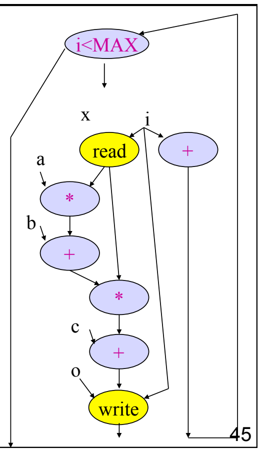
Concerns

- Parallelism in hyperblock
  - Especially if memory sequentialized
    - Disambiguate memories?
    - Allow multiple memory banks?
- Only one hyperblock active at a time
  - Share hardware between blocks?
- Data only used from one side of mux
  - Share hardware between sides?
- Most logic in hyperblock idle?
  - Couldn't we pipeline execution?

Pipelining

for (i=0;i<MAX;i++)
  o[i]=(a*x[i]+b)*x[i]+c;

- If know memory operations independent



Summary

- Language (here C) defines meaning of operations
- Dataflow connection of computations
- Sequential precedents constraints to preserve
- Create basic blocks
- Link together
- Merge into hyperblocks with if-conversion
- Result is logic and registers → RTL

Admin

- Assignment 5 out today
- Assignments 3, 4 graded
- Reading for Wednesday online
- Office hour tomorrow (Tuesday)
  - 5:40pm-6:30pm

Big Ideas:

- Semantics
- Dataflow
- Mux-conversion
- Specialization
- Common-case optimization