

1) Numbers: 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Hex: ABC = A.16<sup>2</sup> + B.16 + C

16<sup>0</sup> = 1, 16<sup>1</sup> = 16, 16<sup>2</sup> = 256

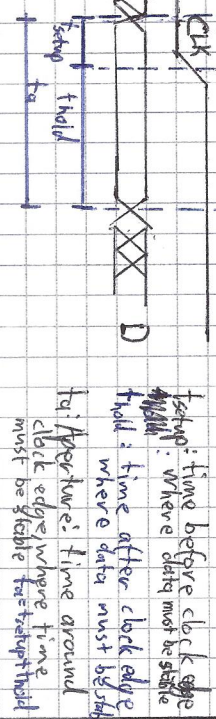
16<sup>3</sup> = 4096, 16<sup>4</sup> = 65536, 16<sup>5</sup> = 1048576

0-9, A=10, B=11, C=12, D=13, E=14, F=15

Unsigned: num positive  
 ↳ Range: 0-2<sup>n</sup>  
 ↳ Sign-Magnitude: exists bit 0=+, 1=-  
 ↳ Variable: for num in bit unreserved  
 ↳ packed: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648, 4294967296, 8589934592, 17179869184, 34359738368, 68719476736, 137438953472, 274877906944, 549755813888, 1099511627776, 2199023255552, 4398046511104, 8796093022208, 17592186044416, 35184372088832, 70368744177664, 140737488355328, 281474976710656, 562949953421312, 1125899906842624, 2251799813685248, 4503599627370496, 9007199254740992, 18014398509481984, 36028797018963968, 72057594037927936, 144115188075855872, 288230376151711744, 576460752303423488, 1152921504606846976, 2305843009213693952, 4611686018427387904, 9223372036854775808, 18446744073709551616, 36893488147419103232, 73786976294838206464, 147573952589676412928, 295147905179352825856, 590295810358705651712, 1180591620717411303424, 2361183241434822606848, 4722366482869645213696, 9444732965739290427392, 18889465931478580854784, 37778931862957161709568, 75557863725914323419136, 151115727451828646838272, 302231454903657293677544, 604462909807314587355088, 1208925819614629174710176, 2417851639229258349420352, 4835703278458516698840704, 9671406556917033397681408, 19342813113834066795362816, 38685626227668133590725632, 77371252455336267181451264, 154742504910672534362902528, 309485009821345068725805056, 618970019642690137451610112, 1237940039285380274903220224, 2475880078570760549806440448, 4951760157141521099612880896, 9903520314283042199225761792, 19807040628566084398451523584, 39614081257132168796903047168, 79228162514264337593806094336, 158456325028528675187612188672, 316912650057057350375224377344, 633825300114114700750448754688, 1267650600228229401500897509376, 2535301200456458803001795018752, 5070602400912917606003590037504, 10141204801825835212007180075008, 20282409603651670424014360150016, 40564819207303340848028720300032, 81129638414606681696057440600064, 162259276829213363392114881200128, 324518553658426726784229762400256, 649037107316853453568459524800512, 1298074214633706907136919049601024, 2596148429267413814273838099202048, 5192296858534827628547676198404096, 10384593717069655257095352396808192, 20769187434139310514190704793616384, 41538374868278621028381409587232768, 83076749736557242056762819174465536, 166153499473114484113525638348931072, 332306998946228968227051276697862144, 664613997892457936454102553395724288, 1329227995784915872908205106791448576, 2658455991569831745816410213782897152, 5316911983139663491632820427565794304, 10633823966279326983265640855131588608, 21267647932558653966531281710263177216, 42535295865117307933062563420526354432, 85070591730234615866125126841052708672, 170141183460469231732250253682105417344, 340282366920938463464500507364210834688, 680564733841876926929001014728421669376, 1361129467683753853858002029456843338752, 2722258935367507707716004058913686677504, 5444517870735015415432008117827373355008, 10889035741470030830864016235654746710016, 21778071482940061661728032471309493420032, 43556142965880123323456064942618986840064, 87112285931760246646912129885237973680128, 174224571863520493293824259770475947360256, 348449143727040986587648519540951894720512, 696898287454081973175297039081903789441020224, 1393796574908163946350594078163807578882044448, 2787593149816327892701188156327615157768888896, 5575186299632655785402376312655230315557777792, 11150372599265311570804752625310460631115555584, 22300745198530623141609505250620921262231111168, 44601490397061246283219010501241842524462222336, 89202980794122492566438021002483685048924444672, 178405961588244985132876042004967370097888889344, 356811923176489970265752084009934740195777778688, 71362384635297994053150416801986948039155557376, 142724769270595988106300833603973896078311114752, 285449538541191976212601667207947792156622229504, 570899077082383952425203334415895582313244459008, 1141798154164767904850406668831791164626488918016, 2283596308329535809700813337663582329252977836032, 4567192616659071619401626675327164658505955672064, 9134385233318143238803253350654329317011911344128, 18268770466636286477606506701308658634023822688256, 36537540933272572955213013402617317268047645376512, 73075081866545145910426026805234634536095290753024, 146150163733090291820852053610469269072190581506048, 292300327466180583641704107220938538144381163012096, 584600654932361167283408214441877076288762326024192, 1169201309864722334566816428883754152577524652048384, 2338402619729444673133632857767508305155049304096768, 4676805239458889346267265715535016610310098608193536, 9353610478917778692534531431070033220620197216387072, 1870722095783555738506906286214006644124038435430416, 3741444191567111477013812572428013288488070860860832, 7482888383134222954027625144856026576976141721721664, 149657767662684459080552502897120531539522834434432128, 299315535325368918161105005794241063079045668868864384, 598631070650737836322210011584482126158091337737728768, 1197262141301475672644420023168964252716182675475557504, 2394524282602951345288840046337928505432365350951115008, 4789048565205902690577680092675857010864730701902230016, 9578097130411805381155360185351714021729461403804460032, 1915619426082361076231072037070342804345890280880892064, 3831238852164722152462144074140685608691781761761784128, 7662477704329444304924288148281371217383563523523568352, 15324955408658888609848576296562742346767127047047136704, 30649910817317777219697152593125484693534254094094273408, 61299821634635554439394305186250969387068508188188746816, 122599643269271108878788610372501938774137016376377513632, 245199286538542217757577220745003877548274032752755027264, 490398573077084435515154441490007755096548065505510054528, 980797146154168871030308882980015510193096131011020109056, 1961594292308337742060617765960031020386192262022040218112, 3923188584616675484121235531920062040772384524044080436224, 7846377169233350968242471063840124015444768048088172448, 156927543384667019364849421276802480308895360961763456, 3138550867693340387296988425536049606177907219235271104, 6277101735386680774593976851072099213558014438705542208, 12554203470773361549187953702144198610116087777411084416, 251084069415467230983759074042883972202321735548228288, 502168138830934461967518148085767844404643091096456576, 1004336277661868923935036296171535688809286182192913152, 2008672555323737847870072592343071377618572364385826304, 4017345110647475695740145184686142755237144728671652608, 8034690221294951391480290369372285510474297557343305216, 1606938044258990278296058073874457102094859511466601032, 3213876088517980556592116147748812441889719022933202064, 6427752177035961113184232295497648879779438045866404128, 12855504354071922226368465590995297759558876091732808384, 25711008708143844452736931181990595519117717383465616672, 5142201741628768890547386236398119023823543476693133144, 10284403483257537781094772472796238047647086953386266288, 20568806966515075562189544945592476095294173906772532576, 41137613933030151124379089891184952190588367813545165152, 8227522786606030224875817978236990438117673562709033024, 16455045573212060449751635956473980876235347125066066048, 32910091146424120899503271912947961752470694250132132096, 65820182292848241799006543825895923504941388500264264192, 131640364585696483598013087651791847009882777000528528384, 263280729171392967196026175303583694019765554001057656768, 526561458342785934392052350607167388039531108002115113536, 1053122916685771868784104701214334776079062216004223027072, 2106245833371543737568209402428669552158124432008446054144, 4212491666743087475136418804857339116236248864016892108288, 8424983333486174950272837609714678324472497728033784216576, 16849966666972349900545675219429346648944995456067568433152, 3369993333394469980109135043885869329788999091213511366624, 6739986666788939960218270087771738659577998182427022733248, 13479973333577879920436540175543477319155996364854045466496, 26959946667155759840873080351086954638311992729708090932992, 53919893334311519681746160702173909276623985459416181865984, 107839786668623039363492321404347818553247970918832363731968, 215679573337246078726984642808695637106495941837664727463936, 431359146674492157453969285617391274212991883675329454927872, 862718293348984314907938571234782548425983767350658909755744, 1725436586697968629815877142569565096851967534701317819511488, 3450873173395937259631754285139130193703935069402635639022976, 6901746346791874519263508570278260387407870138805271278045952, 13803492693583749038527017140556520774815740277610542556091904, 27606985387167498077054034281113041549631480555221085112183808, 55213970774334996154108068562226083099262961110442162243667616, 110427941548669992308216137124452166198525922220884284535335232, 220855883097339984616432274248904332397051844441768569070670464, 441711766194679969232864548497808664794103688883537138141340928, 883423532389359938465729096995617329588207377767074276282681856, 1766847064778719876931458193991234659176414755534148552565363712, 3533694129557439753862916387982469318352829511068297105130727424, 7067388259114879507725832775964938636705659022136594210261454848, 14134776518229759015451665551929877273411318044273188420522909696, 28269553036459518030903331103859754546822636088546376437055819392, 56539106072919036061806662207719509093645272177092752874111638784, 113078212145838072123613324415439018187290544344355505748223277568, 226156424291676144247226648830878036374581088688711111494456555136, 45231284858335228849445329766175675274916217737742223008911211230272, 90462569716670457698890659532351350549832435475484446017822462464, 180925139433340915397781319064702701099664870950968892035644924928, 361850278866681830795562638129405402199329741901937784071289849856, 723700557733363661591125276258810804398659483803875568142579699712, 144740111546672732318225055251762160879731896760775113635939939424, 289480223093345464636450110503524321759463793521550227271879878848, 578960446186690929272900221007048643518927587043100454543759757696, 1157920892373381858545800442014097287037855174086200909087519515392, 2315841784746763717091600884028194574075710348172401818175039030784, 4631683569493527434183201768056389148151420696344803636350078061568, 9263367138987054868366403536112778296302841392689607272700156123136, 18526734277974109736732807072225596592605682785379214544002312246272, 37053468555948219473465614144451193185211365570758429088004624492544, 74106937111896438946931228288902386370422731141516858176009249985088, 148213874223792877893862456577804772740845462283033716018498499970176, 296427748447585755787724913155609545481690925766067432036996999840352, 592855496895171511575449826311219090963381851532134864073993999680704, 1185710993790343023150899652622438181926763703064269728147987999361408, 2371421987580686046301799305244876363853527406128539456295975996828016, 4742843975161372092603598610489752727707054812257078912591951993656032, 9485687950322744185207197220979505455414109624514157825183903987312064, 18971375900645488370414394441959010910828219249028315653677807974624128, 37942751801290976740828788883918021821656438498056631307355615949248256, 75885503602581953481657577767836043643312876996113262614711231898496512, 151771007205163906963315155535672087286625753992226525229422463796993024, 303542014410327813926630311071344174573251507984453050458844927593986048, 607084028820655627853260622142688349146503015968906100917689855187972096, 1214168057641311255706521244285376698293006031937812201835379710375944192, 2428336115282622511413042488570753396586012063875624403670759420751888384, 4856672230565245022826084977141506793172024127751248807341518841503776768, 9713344461130490045652169954283013582344048255502497614623037683007553536, 1942668892226098009130433890856602716468809651100499522



# Input timing constraints



# Output timing constraints



**Clock Skew**: Difference in clock between two registers

**CLK1**:  $t_{skew}$ : Clock Skew

**CLK2**

**Timing**

**CLK**:  $t_{setup}$ ,  $t_{hold}$ ,  $t_{prop}$

**CLK**:  $t_{prop} + t_{pd} + t_{setup}$ ,  $t_{pd} > t_{hold} - t_{prop}$

**5) MIPS: Microarch (RISC-Type Microcontroller)**

**R-Type Register Type**,  $add\ rd\ rs\ rt \Rightarrow rd = rs + rt$

OP	rs	rt	rd	shamt	func
6bit	5bit	5bit	5bit	5bit	6bit
OP	rs	rt	rd	shamt	func
6bit	5bit	5bit	5bit	5bit	6bit

**I-Type: Immediate Type**,  $addi\ rd\ rs\ imm$

**J-Type: Jump Type**,  $j\ rd\ j\ al$

**OP**:  $addi$ ,  $add$ ,  $j\ rd\ j\ al$ ,  $sind$ ,  $einz$ ,  $j\ j\ type$

**Memory**: byte addressable memory

Word Address	Data	Word
0x4	F2	F1
0x8	F2	F1
0xC	F2	F1
0x0	F2	F1

**Registers**:  $MSB \rightarrow 1$ ,  $LSB \rightarrow 31$

Register Name	Register No.	Usage	Reserved
\$0	0	constant val 0	Reserved
\$1	1	assembler temp	Reserved
\$2	2	return values	Reserved
\$3	3	arguments	Reserved
\$4	4	temp	Reserved
\$5	5	temp	Reserved
\$6	6	temp	Reserved
\$7	7	temp	Reserved
\$8	8	temp	Reserved
\$9	9	temp	Reserved
\$10	10	temp	Reserved
\$11	11	temp	Reserved
\$12	12	temp	Reserved
\$13	13	temp	Reserved
\$14	14	temp	Reserved
\$15	15	temp	Reserved
\$16	16	temp	Reserved
\$17	17	temp	Reserved
\$18	18	temp	Reserved
\$19	19	temp	Reserved
\$20	20	temp	Reserved
\$21	21	temp	Reserved
\$22	22	temp	Reserved
\$23	23	temp	Reserved
\$24	24	temp	Reserved
\$25	25	temp	Reserved
\$26	26	temp	Reserved
\$27	27	temp	Reserved
\$28	28	temp	Reserved
\$29	29	temp	Reserved
\$30	30	temp	Reserved
\$31	31	temp	Reserved

**MIPS PC starts at 0x00400000**

**MIPS Memory Layout**:  $max. 2^{32} = 4GB$  from 0x00000000 to 0xFFFFFFFF

Reserved	Dynamic Data	Static Data	Text	Reserved
0x00000000 - 0x0000000F	0x00000010 - 0x000000FF	0x00000100 - 0x00000FFF	0x00001000 - 0x00001FFF	0x00002000 - 0x00002FFF

**6) Processor**:  $Single\ Cycle$ ,  $Multi\ Cycle$

**Single Cycle**:  $Each\ instr. takes\ 1\ cycle$

**Multi Cycle**:  $Each\ instr. takes\ 4\ cycles$

**Control Store**:  $Stores\ control\ signals\ for\ all\ possible\ states$

**Pipeline**: use hardware better

**Steady State**:  $Pipeline\ is\ full$

**Setup**:  $Optimal = 1/(C/T + S)$

**Cost**:  $increase \rightarrow additional\ HW$

**Instruction Cycle**:  $(A) Instr. Fetch (IF)$

**Decode**:  $2) Instr. Decode \& Register Operands$

**Execute**:  $3) Execute/Calculate Mem. Addr. (EX/ALU)$

**Memory**:  $4) Mem. Operands Fetch (MEM)$

**Store**:  $5) Store/Write back Result (WB)$

**Problems**:  $Resource\ contention$

**Flow Control**:  $Flow\ dependencies$

**45-937-747**

**Pascal Wacker**



# 7) Tomasulo's Algorithm: Uses Register Alias Table (RAT), Functional Units (FU) and Reservation Stations (RS), RAT & RS have valid flag and value fields

- 1) Decode operation, put operands in RS, if data in RAT is valid use it, otherwise store tag
- 2) If all operands of RS are available pass to FU
- 3) After execution broadcast value from FU to RS and RAT

## 8) Other Execution models:

### - VLIW: Very Long Instruction Word

Some instructions contain several instructions which are executed on separate FU

### - SIMD: Single Instruction Multiple Data

Array Processor: Instruction operates on multiple data elements of the array at once

Vector Processor: Instruction operates on multiple data elements at the same space in consecutive steps

→ dependencies within vector features generate lot of work → slower instr. fetch bandwidth

→ high register memory access

→ works only if parallelism is regular

Control Registers: VLEN, VSTR, VMASK (used for condition)

Vector FU: deeply pipelined & fast CLK, loops are vectorizable if independent

### Vector Chaining: Forwarding for Vectors

## GPU:

- Instruction Pipeline operates like SIMD pipeline, but programming is using threads

### SIMT (Single Instruction Multiple Threads)

↳ Multiple instruction streams of scalar instructions

→ can treat each thread separately

→ can group threads into warps of 32, to maximize SIMD throughput

↳ Warp/leave front: A group of threads executing the same instruction → basically SIMD by HW

↳ Branching in warps: Prediction by GPU using mask (similar to MASK) to only operate on correct data. Done by HW, not like vectors done by SW programmer

↳ Improving SIMD utilisation: find threads at same PC and group them

## Programming vs. Execution Model

Programming Model: how programmer expresses code: Sequential (von Neumann), Data Parallel (SIMD), Data Flow, Multithreads (MIMD, SPMD)

Execution Model: how HW executes:

Out-of-Order, Vector/Array-Processor, Data Flow, Page

→ no need to match, ex: sequential on 0-0-0 Processor

## 9) Memory:

- Flip-Flops and Latches:

+ Fast parallel access - Expensive (20+ transistors/bits)

- Static RAM (SRAM):

+ Relative Fast, only one data word at a time

- still expensive (6 transistors/bit) - ~~SRAM~~

- Dynamic RAM (DRAM)

↳ cheap - slower, reading destroys content (refresh)

1:  $\frac{1}{2} \rightarrow \frac{1}{2}$  one data word, need special process

- SSD, HDD, usw. + cheaper - slower

Memory Banking: Used to resolve long memory latency, All "n" memory banks share one data bus and one address bus.

Fetch from bank 0, bank 1, ..., bank n-1

↳ throughput of 1 Element/Cycle if done correctly

↳ Number of banks  $\geq$  Memory latency (usually  $\geq 2$  banks)

↳ Next bank address = previous bank address + stride

Cache: - Temporal locality, keeps recent data, in smaller but faster memory (higher level) → closer to processor

- Spatial locality, keeps close data in higher level caches (like next word, next and prev entry)

- Hit: entry found in cache

- Miss: entry not found in cache

↳ Compulsory: first time data is accessed

↳ Capacity: cache too small to hold data (write with LRU)

↳ Conflict: data of interest maps to same address

↳ Miss penalty: time it takes to retrieve data of miss

- Miss rate: #misses/#total access =  $1 - \text{HitRate}$  (#hit/#total)

- Average Memory Access Time (AMAT)

↳ AMAT =  $t_{\text{cache}} + \text{MissRate} \cdot \text{MissPenalty}$  ( $\leftarrow \text{VM} = \text{HD}$ )

↳  $t_{\text{cache}} + \text{MR} \cdot \text{cache} \cdot t_{\text{main}} + \text{MR} \cdot t_{\text{main}} \cdot t_{\text{main}} \cdot t_{\text{main}}$

Example: 2000 loads, 1250 in cache

MissRate =  $3500/2000 = 0.375$

HitRate =  $1250/2000 = 0.625$

Example:  $t_{\text{cache}} = 1 \text{ cycle}$ ,  $t_{\text{main}} = 100 \text{ cycles}$ ,  $\text{warp size} = 1$  AMAT

AMAT =  $t_{\text{cache}} + \text{MR} \cdot \text{cache} \cdot t_{\text{main}} = (1 + 0.375 \cdot 100) = 36.5 \text{ cycles}$

## Memory Address: TAG INDEX OFFSET

45-937-717

Pascal Water 3/6

Memory Address:  $B = 2^2 = 4$

(1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Set 1 2 3 4

Block 1 2 3 4

Virtual Memory: Block = Page, Miss = Page Fault

Wie cache, mit!

Tag = Virtual Page Number

Translating Addresses: Page Table is a LookUp-Table (LUT) holding an entry for every virtual page, LUT between virtual and physical address space

10) Some numbers:

$10^6 = \text{MHz}$ ,  $10^9 = \text{GHz}$

Decimal Hex Binary

1 1 1

2 2 2

3 3 3

4 4 4

5 5 5

6 6 6

7 7 7

8 8 8

9 9 9

10 10 10

11 11 11

12 12 12

13 13 13

14 14 14

15 15 15

16 16 16

17 17 17

18 18 18

## Example Cache:

Memory Word (4 bytes, even byte addressable)

Cache Type: Direct Mapped

Cache Size: 8 words

Cache block size: 1 word

Cache access time: 2 Cycles

Main Memory Access time:  $t_{\text{main}} = 40 \text{ Cycles}$

$C = 8 \text{ words} = 4 \text{ bytes} = 32, b = 1 \text{ word} = 4 \text{ bytes}$

$B = C/b = 32/4 = 8, N = 1, S = B = 8$

$\rightarrow S = 8, \log_2(S) = \log_2(8) = 3 \rightarrow \text{index} = 3 \text{ bits}$

Tag

Set 1

Set 2

Set 3

Set 4

Set 5

Set 6

Set 7

Set 8

Set 9

Set 10

Set 11

Set 12

Set 13

Set 14

Set 15

Set 16

Set 17

Set 18

Set 19

Set 20

Set 21



# VLW (Very Long Instruction Word)

Advantages: No need for dynamic scheduling HW  $\rightarrow$  simpler HW  
 + No need for dependency checking within VLW instr.  $\rightarrow$  simple HW for multiple instr. issue + no renaming  
 + No need for instruction alignment/distribution after fetch to different FU  $\rightarrow$  simple HW

Disadvantages: - compiler needs to find N independent operations per cycle  $\rightarrow$  parallelism loss  
 - recompilation required when execution with (N) or instr. latency or FU changes (unlike super scalar processing)  
 - locked execution causes independent operations to stall  $\rightarrow$  No instr. can progress until slowest instr. completes

## SISD: Single Instruction Single Data

SIMD: Single Instruction Multiple Data

MISD: Multiple Instruction Single Data

MIMD: Multiple Instruction Multiple Data

Multi-processor, Multi-headed processor

## Array VLS: Vector Processor

Array: multiple FU, jede bearbeitet ein Array element.

Vector: eine FU, elemente werden einzeln nach dem anderen durchgeschickt, spart  $\rightarrow$  72D!

Wann muss ID nach Fall event ausgesetzt werden?  $\rightarrow$  only if not 0-0-0, 0-0-0 oder again 0-0-0!

Falls processor Resource Contention, ex: add waits for an adder to become free  $\rightarrow$  again 0-0-0!

if Data Dependency no need for ID again!  $\rightarrow$  if Data Dependency no need for ID again!  $\rightarrow$  if Data Dependency no need for ID again!

## In-Order Pipeline mit ROB (Reorder Buffer)

Fetch  
 Decode Access regfile/ROB, allocate entry in ROB, check if can execute and if so dispatch

Execute Instr. instr. can execute 0-0-0!  
 Completion write Result to ROB  
 Retirement/Commit: check for exceptions to retire  $\rightarrow$  if none write result to architec. regfile, free memory  $\rightarrow$  send back pipeline, start at exception handling

## Verilog

module mux2 (input [1:0] i, output s, input z);

assign s = (z ? i[1] : i[0]);

endmodule

module one (input [3:0] data, input sel1, input sel2, output z);

wire [1:0] temp1;

mux2 i0 (data[1:0], sel1, temp1[0]);

mux2 i1 (data[3:2], sel1, temp1[1]);

mux2 final (temp1, sel2, z);

endmodule

Error: input connected with output! better use a connect by name

module two (input [3:0] a, input [0:3] b, output reg [3:0] z);

always @ (\*)

if (a == 0)

z = {b[0], b[1], b[2], b[3]};

else

z = a + b

endmodule

Syntax: OK, could be written as  $z = a + b$ , mixing of C3.0 and C5.3. okay but strange

module three (clk, rst, a, b, c, z);

input a, b, c, clk, rst; output z; reg z;

always @ (\*)

begin

if (a <= a + b;

if (c) a <= ~ (a + b);

always @ (negedge clk)

if (rst) z = 1'b0;

else z = a;

endmodule

Syntax: OK, strange to use non blocking in always..

module four (input [2:0] sel, output reg [5:0] z);

case (sel)

0: z = 6'b00\_0000;

1: z = 6'b00\_0001;

next state <= present;

detected <= 1'b0;

case (present)

INIT: next state <= in ? INIT : ONE;

ONE: if (in)

next state <= INIT;

else

begin

next state <= ONE;

detected <= 1'b1;

end

default: next state <= present

endcase

end

always @ (posedge clk, negedge rst)

if (rst)

present <= INIT;

else

present <= next state

endmodule

OK: Mealy type FSM

module srwm (input [15:0] din, input [23:0] addr,

input w, input clk, output [25:0] dout);

// stuff

endmodule

each address stores 16 bit  $\rightarrow$  16/8 = 2 byte es. 14 addr.

which are 2\*14 = 28. 2\*14 = 28. 2\*14 = 28. 2\*14 = 28.

in total 16k \* 2 bytes = 32k bytes storage

## Systemic Arrays

Goal: design accelerator that has:

- simple, regular design (keep unique parts small and reg)

- high concurrent  $\rightarrow$  fast

- balanced computation and I/O (memory) bandwidth

Idea: Replace single processing element (PE) with regular array of PEs and carefully orchestrate flow of data between the PE

16-937-717  
 Pascal Welter 4/6

- the PE collectively transform Data

Benefit: Maximised computation done on a single piece of data element brought from MEM

Advantages: Multiple use of data items  $\rightarrow$  reduce need for fetching/re-fetching

high concurrent + regular design

Disadvantage: - not good at exploiting irregular parallelism

- special purpose, not generic

Q00 a01 a02 a03

Q04 a05 a06 a07

Q08 a09 a10 a11

Q12 a13 a14 a15

Q16 a17 a18 a19

Q20 a21 a22 a23

Q24 a25 a26 a27

Q28 a29 a30 a31

Q32 a33 a34 a35

Q36 a37 a38 a39

Q40 a41 a42 a43

Q44 a45 a46 a47

Q48 a49 a50 a51

Q52 a53 a54 a55

Q56 a57 a58 a59

Q60 a61 a62 a63

Q64 a65 a66 a67

Q68 a69 a70 a71

Q72 a73 a74 a75

Q76 a77 a78 a79

Q80 a81 a82 a83

Q84 a85 a86 a87

Q88 a89 a90 a91

Q92 a93 a94 a95

Q96 a97 a98 a99

Q100 a101 a102 a103

Q104 a105 a106 a107



# Practical Exercise 18

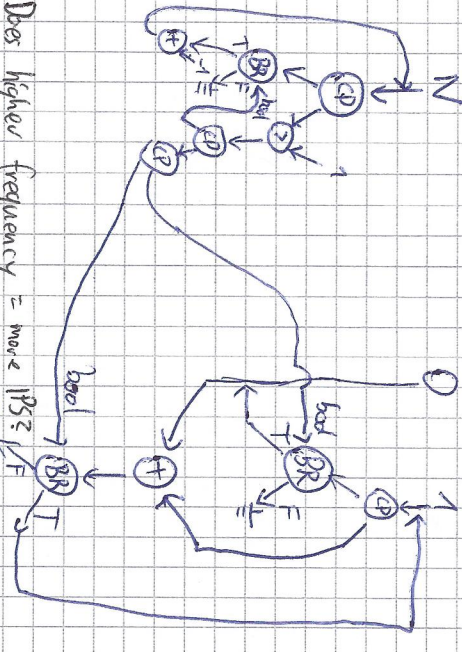
fib: addi \$s0, \$s0, 16 // allocate space  
 sw \$s0, 0(\$s0) // save r16  
 add \$s1, \$s0, 4(\$s0)  
 sw \$s1, 4(\$s0)  
 add \$s1, \$s0, 8(\$s0)  
 sw \$s1, 8(\$s0)  
 addi \$s1, \$s0, 11  
 sw \$s1, 12(\$s0)  
 sw \$s1, 14(\$s0)  
 add \$s2, \$s1, 16(\$s0)

int fib(int n) {  
 int a = 0;  
 int b = 1;  
 int c = a + b;  
 while (n > 1) {  
 c = a + b;  
 a = b;  
 b = c;  
 n--;  
 }  
 return c;  
}

branch: slli \$s3, \$s0, 2  
 bne \$s3, \$s0, done  
 add \$s2, \$s1, 16(\$s0)  
 add \$s1, \$s0, 4(\$s0)  
 addi \$s1, \$s0, 11  
 sw \$s1, 12(\$s0)  
 sw \$s1, 14(\$s0)  
 add \$s2, \$s1, 16(\$s0)  
 jr \$s31

done: lw \$s3, 12(\$s0)  
 lw \$s1, 8(\$s0)  
 lw \$s2, 4(\$s0)  
 lw \$s1, 0(\$s0)  
 addi \$s0, \$s0, 16  
 jr \$s31

## Dataflow version:



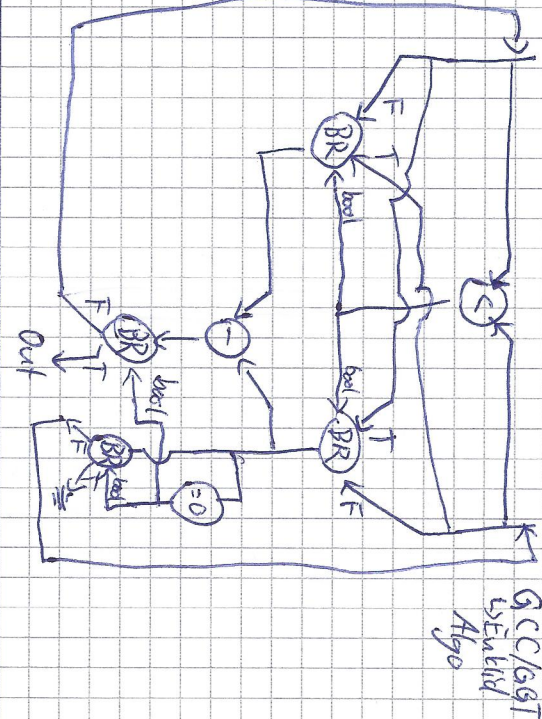
Does higher frequency = more IPS?   
 ↳ No, maybe a slower CPU has higher IPC.   
 ↳ Does higher IPC = finish program faster?   
 ↳ No, maybe higher IPC because smaller instr. → total # of instr. could be different

ISA A: 40 IPC, 500MHz. What is performance?   
 ISA B: 2 IPC, 600MHz in Million Instr. Per Second

A: 10 instr. 500'000'000 cycles = 500 MIPS   
 B: 2 instr. 600'000'000 cycles = 1200 MIPS

We don't know which one is better, since compiled program could use 100 instr on ISA B and 100'000 on ISA A.   
 Difference ISA/Microarchitecture:   
 ISA = exposed to software, transparent to compiler/programmer   
 Machine has no subunit instr. ISA   
 - ALU doesn't have subunit. Microarchitecture   
 - Machine doesn't have condition codes ISA

## EX 3



## Practical EX 30

As pipeline depth increases, the latency for a single instr. increases → each additional pipeline latch adds processing overhead

In a superscalar processor, the hardware detects the data dependencies between concurrently fetched instr. In contrast in a VLIW processor, the compiler does TLB (Translation Lookaside Buffer) caches: Page table entries

For a vector length of N we need N FU in an array processor and 1 in a vector processor

Max PS: # threads / # threads (max P) ex: 1024 / 64 threads = 16 threads   
 Max PS = 20 / 2 = 10 threads

8 FU, each with own broadcast bus. 15-937-717   
 32 x 64-bit architectural registers. Local Waker 5/6   
 16 reservation station entries per FU   
 Each RS has 2 source registers. 8FU 2 source regist

# of log comparators per RS: 8:2   
 # of total comparators: 8 \* 16 \* 8 \* 2 + 32 \* 8 = 8FU   
 min size of log: log(16 \* 8) = 7   
 16/8FU 8FU

min size of RAT/Forward Register File: 720 \* 32 (72 = 64 bit data + 7 bit tag + 1 bit valid)   
 min possible size of tag storage in Machine: 7 \* 32 + 7 \* 16 \* 8 \* 2 = 2 source registers   
 32 registers 1/8 RS/FU 8FU

## 20 = 1

1: 2	11 2 048	21 2 097 152
2: 4	12 4 096	22 4 194 304
3: 8	13 8 192	23 8 388 608
4: 16	14 16 384	24 16 777 216
5: 32	15 32 768	25 32 155 432
6: 64	16 65 536	26 64 310 864
7: 128	17 131 072	27 128 621 728
8: 256	18 262 144	28 256 1243 456
9: 512	19 524 288	29 512 2486 912
10: 1024	20 1048 576	30 1024 4973 824

Interlocking: each reg in reg file with instr. writing to reg file + valid bit   
 - instr. decode check for valid bit   
 - if not stall   
 + simple - stalls for all dependencies   
 Combinational Dependency Checking Logic   
 if instr. in later stage writes to any source   
 stall   
 ① only stalls flow - complex specially with deep pip   
 Forwarding/Bypass   
 + less stalling - skipping Reg file