Bonus Articles

This is a curated list of keywords and their meaning for CS21, arranged per lecture video. Links to articles are provided at the end.

Lecture 1 : Automotive ECUs

ECU - Electronic Control Unit

MCU - Micro Controller Unit

ADAS - Advanced Drive-Assistance Systems

LIN - Local Interconnect Network

DSPI - Deserial Serial Peripheral Interface

CAN-FD - Control Area Network - Flexible Data

MIPS - Microprocessor without Interlocking Pipeline Stages

RISC - Reduced Instruction Set Computer

SENT - Single Edge Nibble Transmission

DSC - Digital Signal Control

SPC5 32-Bit Automotive MCU

AVR MCU - Advanced Virtual RISC MCU

PIC MCU - Peripheral Interface Controller MCU

SPI - Serial Peripheral Interface

Article Items

SPC5 32-Bit Automotive MCU

Manufacturer: STMicroelectronics

Memory: 128 KB - 10 MB (flash)

Clock Speed: 48 MHz - 200 MHz

Cores: 3

Temperature Tolerance: -40 °C to 165 °C

Additional Features: Can endure heavy shocks

Common Uses: engine and chassis control, ADAS (see meaning above), driver safety, windows, doors

MIPS32 and MIPS64 Architecture

Manufacturer: N/A

Memory: N/A

Clock Speed: N/A

Cores: N/A

Temperature Tolerance: N/A

Additional Features: Code-efficient, low power consumption, low cost

Common Uses: electronic doors, hoods, windshield wipers, automatic windows, chassis and powertrain electronic systems, infotainment system, ADAS, and autonomous driving

16-Bit MCUs for Engine ECUs

Manufacturer: Microchip Technology, Inc.

Memory: N/A

Clock Speed: N/A

Cores: N/A

Temperature Tolerance: -40 °C to 150 °C

Additional Features: withstands sudden shocks, high-speed mobility, oil and grease deposition

Common Uses: controlling motors, actuators, turbocharger wastegates, EGR valves, and oil/water pumps

8-Bit MCUs (PIC and AVR MCUs)

Manufacturer: Microchip Technology, Inc.

Memory: N/A

Clock Speed: N/A

Cores: N/A

Temperature Tolerance: -40 °C to 150 °C

Additional Features: low power consumption, contain CIPs (Core-Independent Peripherals)

Common Uses: small program codes, control analog sensors, digital sensors, capacitive touch functionalities, LED lighting systems

S32K Automotive MCUs

Manufacturer: NXP Semiconductors N.V.

Memory: 128 kB to 8 MB (flash)

Clock Speed: N/A

Cores: 1,2,lockstep

Temperature Tolerance: -40 °C to 150 °C

Additional Features: designed on Arm Cortex-M series RISC architecture, AEC-Q100 Grade 0, Grade 1, and Grade 2 certifications, pin configurations: 32 to 176 pins and 48 to 289 pins

Common Uses: body, electrical unit, and zone controls for car ECUs

32-Bit AURIX TriCore Microcontroller

Manufacturer: Infineon Technologies AG

Memory: 0.5 MB to up to 16 MB (flash)

Clock Speed: 133 MHz to 300 MHz

Cores: 3

Temperature Tolerance: -40 °C to 150 °C

Additional Features: with dedicated SRAM; supports CAN, LIN, FlexRay, and SPI protocols

Common Uses: control the internal combustion engine, transmission control units, and electric power steering systems; ADAS, autonomous driving control, drive safety management, and in-car connected services automation

Source: [What CPU Does a Car ECU Run On?]

Lecture 2 : MIPS Goes Open Source

Terms

SIMD - Single Instruction, Multiple Data

MIPS - Microprocessor without Interlocking Pipeline Stages

RISC-V - Reduced Instruction Set Computer - V

DSP - Digital Signal Processing

Key Insights

1. Wave Technologies announced on Dec 17, 2018 that MIPS will be on open-source
2. MIPS R6 core will be available by 2019
3. MIPS is more complete than RISC-V due to its DSP and SIMD capabilities
4. MIPS Open Program will give participants access to latest 32-bit and 64-bit versions, but still no remarks on who will manage it
5. MIPS IPs on AI through its plan "AI for All" will be its strategy to the difficulties from licensing revenue in open source
6. MIPS entered into a separate agreement to sell patents to Bridge Crossing

7. China led several generations of MIPS-based Godson chips

Source : MIPS Goes Open Source

Lecture 3 : The History, Controversy, and Evolution of the Goto statement

History of Goto

Timeline

1. Mark I changed tape measures as jump instruction, used whole addresses as target addresses
2. ENIAC used vacuum tubes instead of tape measures for jumps, used whole addresses as target addresses
3. EDSAC introduced conditional jump instruction through comparing program counter with predecessor (the difference)
4. Fortran programming language introduced "goto", "if" and "computed goto" which is goto using a variable value, similar to jump tables
5. ALGOL 60 kept goto and jumps could either be numbers or identifiers. It also added a for loop capability and code blocks. Blocks allowed multiple statements grouped together with the same condition, replacing the goto with "if". Switch replaced the computed goto of Fortran.
6. Pascal kept the goto, but replaced the switch with case labels. Also, Pascal restricted numbers to labels.
7. C used if, else, and goto while replacing the case labels with the switch statement (with added fall-through).

Evolution

Jump Tables

1. Assembly Jump Tables

```
.word Line0,Line1,Line2,default <- jump table definition
Line0:   Todo
j   finish
Line1:   Todo
j   userInteraction
Line2:   Todo
j   userInteraction
default:   Todo
j   userInteraction
```

2. Computed Goto

```
IF (VARIABLE VALUE) THEN GOTO 110,120,130,140
```

(disclaimer: the numbers are line numbers in code)

3. ALGOL switch

```
switch s = label1, label2, label3;
# ...
goto s[i];
```

4. Fortran Case

```
case i of:
    1:
    2:
    3:
```

5. C switch

```
switch(i){
    case 1:
    case 2:
    case 3:
}
```

Conditional Jump for skipping stuff

1. Conditional Jump

```
IF ... THEN GOTO 50
50: ...
```

2. If statement

```
if(condition){
    \\ statement goes here
}
```

Conditional Jump for looping

1. Conditional Jump

```
30: IF ... THEN GOTO 50
50: ... IF ... THEN GOTO 30
```

Turned to its direct successors: for, while, do/while loops

2. For loop

```
for(initialization,condition,increment){
    \\ statements
}
```

3. While loop

```
while(condition){
    \\ statements
}
```

4. Do-while loop

```
do{
    \\ statements
}while(condition);
```

Jumping to an error handler

1. Jump on Error Handler

```
ON ERROR GOTO 9999
```

2. Exception blocks

```
try {
...
throw(...);
...
} catch(...) {
...
}
```

Jumping out of the loop

1. Conditional Jump

```
FOR I = 1 to 10
...
IF ... THEN GOTO 10
```

```
...
NEXT I
10 ...
```

2. Break/Labeled Break Statements

```
for(i = 0; i < 10; i++) {
if(...) break;
}
LOOP:
for(i = 0; i < 10; i++) {
if(...) break LOOP;
}
```

Controversy

1. In 1962, Peter Naur deems that built-in syntaxes for if and for loops must be used rather than gotos after observations that some gotos are in-disguise compounds if statements and loops
2. In 1968, Edgar Djikstra called for the abolition of the goto statement
3. Donald Knuth disagrees with Djikstra, calling for proper use of gotos to improve readability and ease of understanding
4. Djikstra conceded that some of Knuth's gotos in his book Structured Programming with Goto Statements are appropriate

Common Uses and Best Practices

1. Soloway, Bonar, and Ehrlich found that students solve better using a break statement than without, pointing to the ease and efficiency of the read/process loop over the process/read loop in solving problems.
2. Goto statements allow for the removal of recursion while keeping its general recursive structure
3. Goto labels make clearer expectations to the programmer of codes running only once in retry vs doing it in a for loop, which expects the code may run multiple times
4. Martin Hopkins claims goto statements are a middle ground of assembly and high level programming, giving optimizations without sacrificing portability.
5. Donald Knuth proved that without multi-level break statements, some gotos cannot be eliminated without losing efficiency
6. Tail recursion optimization
7. Coroutines
8. Implementing algorithms already expressed in a flowchart
9. Breaking out of nested loops without a multi-level break (like C).

Source : [The History, Controversy, and Evolution of the Goto statement]

Lecture 4 : Special Functions

Key Insights

- It is a function giving particularly good values of interest within a specific use case
- It organizes mathematical calculations
- It functions as a lookup table for desirable numbers
- It is good when it has a few arguments, has rational coefficients when expressed in power series, and has little relations with other functions, and some particular set of problems are reducible to it

Source: Special Functions

Lecture 5: Abridged History of C

Key Insights

- C is born of success and failure
- Multics paved the way for Unix
- CPL (Cambridge Programming Language) paved the way for C
- C was written for Unix, and Unix was written to make programs
- One must write a programming language for both theoretical and practical applications

Source: Origins of C

Lecture 6

Key Insights

- Compare modern instruction set architectures in all three factors: process node, implementation, and ISA
- In implementation-centric view of architecture, implementation of CPU matters more than its instruction set
- In ISA-centric view of architecture, instruction set type (RISC or CISC) matters more than its implementation

Source: RISC vs CISC

Lecture 12: The Microprogramming of Pipelined Processors

Terms

- Pipelined processor - a processor where instructions are handled in different stages
- Microprogramming - a programming paradigm using microinstructions through a pipelined processor
- Micro-order - an instruction in microprogrammed processor which tells a pipeline what to do or what the inputs must be in a register
- Microassembler - a special assembler which maintains a bijection between the mnemonic and the microinstruction (Time-Stationarity) and may keep several copies of the instruction through runtime (Data Stationarity)
- Time-Stationarity - a property where microinstructions are defined at a particular fixed time
- Data-Stationarity - a property where microinstruction are defined at a particular fixed data flow

Key Insights

- There are two extreme methods of microprogramming a pipelined processor: pure Time Stationary and pure Data Stationary.
- Both methods struggle when microprogrammed branches are taken
- A flexible microassembler can introduce the advantages of Data-Stationarity into Time-Stationarity
- Tradeoffs of using Data Stationarity and Time Stationarity in certain sections of the pipeline must be made in complex pipelines where more than one source of data flow occurs.

Source: Microprogramming of Pipelined Processors

Lecture 13: Techniques for Solving Data Hazards

Key Insights

- There are two common ways to solve Data Hazards in a MIPS32 processor: data forwarding and stalling
- An efficient memory hierarchy (cache-RAM-disk design) greatly enhances a microprocessor's performance
- Additional inbound memory can be used to resolve data hazards. When instruction encounters a hazard, it can utilize memory latency to resolve them without using forwarding and stalls, saving power and improving performance.

Sources:

A Comprehensive Analysis on Data Hazard for RISC32 5-Stage Pipeline Processor

Design Example of Useful Memory Latency for Developing a Hazard Preventive Pipeline High-Performance Embedded-Microprocessor

Lecture 14: Control Hazards in Microprocessors